UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Aplicación web para gestionar el uso de abonos y productos fitosanitarios en parcelas agrícolas

Autor/es

ALICIA BERIAIN GIL

Director/es

JESÚS MARÍA ARANSAY AZOFRA

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Aplicación web para gestionar el uso de abonos y productos fitosanitarios en parcelas agrícolas

Alumna:

Alicia Beriain Gil

Tutor:

Jesús María Aransay Azofra

Departamento de Matemáticas y Computación

Logroño, 07,2017

Contenido

RESUMEN

En este trabajo de fin de grado he realizado una aplicación web que se va a emplear en el ámbito agrario. La finalidad de la aplicación es permitir a los agricultores calcular la cantidad de abonos que deben utilizar en su finca. Para ello, los agricultores podrán administrar, entre otras cosas, explotaciones, fincas, cultivos, abonos minerales, abonos orgánicos…

El cliente de esta aplicación ha sido un estudiante de Ingeniería Agrícola de la Universidad de La Rioja.

ABSTRACT

In this project I have developed a web application that is going to be used in the agricultural field. The aim of this application is to allow farmers to work out the quantity of fertilizer they should use in their plantation. In order to do that farmers can manage, amongst others, farmlands, plantations, cultivations, mineral fertilizers, organic fertilizers and so on.

The client of this application has been a student of the agricultural engineering degree of the University of La Rioja.

CAPÍTULO 1. PLANIFICACIÓN

INTRODUCCIÓN

En este TFG se va a trabajar sobre un tema relacionado con la ingeniería agrónoma. En concreto se va a abordar el tema de qué abonos deberá añadir el agricultor a su finca y en qué cantidades.

Una finca tiene en la composición química de su suelo una cantidad de nitrógeno (N), fósforo (P) y potasio (K). Cuando se planta un cultivo en una finca este absorbe parte de estos nutrientes. Para que el cultivo plantado en la finca proporcione el rendimiento deseado, esta falta de nutrientes se debe solventar mediante el abonado de la tierra. El rendimiento de una finca es la cantidad de kilos de cosecha que produce.

Normalmente los agricultores hacen dos abonados. Uno de fondo, antes de plantar el cultivo, y otro de cobertera, después de plantar el cultivo. La aplicación se encargará de decirle al agricultor la cantidad de abono que deberá añadir.

Es importante señalar que en la agricultura hay dos tipos de abonos principales: los abonos orgánicos, que son aquellos de origen animal y los abonos minerales que han sido obtenidos mediante procesos químicos.

Para que la aplicación pueda realizar sus cálculos necesita que el agricultor introduzca varios datos. Lo primero que necesita es que el agricultor introduzca su finca con sus características. Usualmente los agricultores agrupan un conjunto de fincas en explotaciones.

También necesita que el agricultor indique el cultivo que se ha plantado o se va a plantar en esa finca. Posteriormente, también necesitará que el agricultor le indique qué abono se ha añadido de fondo y qué cantidad. A continuación, necesitará que el agricultor introduzca el rendimiento que espera para ese cultivo en esa finca (es decir, los kg de cosecha que se pretenden obtener por hectárea de suelo). Dependiendo del rendimiento que se espera de estos cultivos la cantidad de abonado que hay que añadir será mayor o menor.

Finalmente, la aplicación deberá saber qué tipo de abono desea el agricultor añadir de cobertera para poder indicarle la cantidad que debe añadir de este.

ACTORES

Los actores implicados en este Trabajo Fin de Grado son los siguientes:

- Jesús Aransay: tutor académico del TFG. Profesor del Departamento de Matemáticas y Computación de la Universidad de La Rioja.
- Josu Lana: cliente de la aplicación. Alumno del grado de Ingeniería Agrícola que está realizando también el TFG.
- Juan José Barrio: cliente de la aplicación y tutor de Josu Lana.
- Alicia Beriain: alumna del Grado en Ingeniería Informática y autora del TFG.

A partir de ahora, siempre que nos refiramos al cliente nos estaremos refiriendo a Josu Lana que es en este caso el que actúa con este rol. Aunque tenga un cliente final la aplicación que voy a realizar va a ser genérica para cualquier finca, cultivo, tipo de abono…

CALENDARIO DE TRABAJO

En la siguiente figura podemos observar el calendario de trabajo que se va a seguir para la realización del TFG. Los días que he marcado en rojo son aquellos en los que no voy a trabajar. En la semana 17 y 18 de mayo no he planificado avanzar con el TFG, ya que en esas semanas tengo exámenes y me quiero centrar en ellos. Quitando esas dos semanas calculo que le dedicaré al TFG unas 19 semanas y unos 114 días. Por lo tanto, cada día le tendré que dedicar un poco menos que tres horas. No pretendo programar hasta el día anterior de la entrega del TFG, pero marco esas semanas por si acaso surge algún imprevisto.

febrero

	l	m	m	j	v	s	d
s1			1	2	3	4	5
s2	6	7	8	9	10	11	12
s3	13	14	15	16	17	18	19
s4	20	21	22	23	24	25	26
s5	27	28					

marzo

	l	m	m	j	v	s	d
s5			1	2	3	4	5
s6	6	7	8	9	10	11	12
s7	13	14	15	16	17	18	19
s8	20	21	22	23	24	25	26
s9	27	28	29	30	31		

abril

	l	m	m	j	v	s	d
s9						1	2
s10	3	4	5	6	7	8	9
s11	10	11	12	13	14	15	16
s12	17	18	19	20	21	22	23
s13	24	25	26	27	28	29	30

mayo

	l	m	m	j	v	s	d
s14	1	2	3	4	5	6	7
s15	8	9	10	11	12	13	14
s16	15	16	17	18	19	20	21
s17	22	23	24	25	26	27	28
s18	29	30	31				

junio

	l	m	m	j	v	s	d
s18				1	2	3	4
s19	5	6	7	8	9	10	11
s20	12	13	14	15	16	17	18
s21	19	20	21	22	23	24	25
	26	27	28	29	30		

ANÁLISIS DE REQUISITOS

REQUISITOS FUNCIONALES

En las primeras reuniones (véase acta 3-2 y 8-2) obtuvimos una captura de requisitos inicial con el cliente. Es importante decir que también ha sido ligeramente modificada en las siguientes reuniones.

1. El cliente desea realizar una aplicación web para permitir a los agricultores calcular la cantidad de abono que necesitan en sus fincas o explotaciones agrarias.

2. Un **agricultor** se dará de alta aportando un nombre de usuario y una contraseña, también dará su nombre.

3. Una vez dado de alta el agricultor podrá añadir los datos de una o varias **fincas** a la aplicación.
   - Para ello seleccionará la provincia en la que se encuentra la finca. Además, aportará el número del municipio, polígono y parcela asociada a la finca. Mediante la referencia catastral, accediendo a una API del catastro, la aplicación obtendrá la superficie de la finca y su nombre. El agricultor podrá ser capaz de modificar el nombre de la finca.
   - El agricultor deberá añadir la composición de nitrógeno, potasio y fósforo de la tierra medido en mg/Kg. Opcionalmente también puede añadir la composición de ph, calcio, magnesio y unos extras que describen otras características de las fincas.
   - El agricultor también deberá añadir la textura; densidad de la tierra, medida en gramos por decímetro cúbico (kg/m^3); la materia orgánica del terreno, medida en % respecto a la materia total; la relación carbono nitrógeno (c/n), conductividad y la Capacidad de Intercambio Catiónico (CIC). Por último, debe indicar si es vulnerable o no al nitrógeno.
   - Finalmente, el agricultor indicará si la finca es de regadío o secano. En el caso de que sea de secano tendrá una tasa de mineralización de 1.4, si es de regadío será de 2. La tasa de mineralización es la relación entre la materia orgánica y la materia mineral inerte. Es decir, es el ratio que define la velocidad a la cual la materia orgánica pasa a ser descompuesta por los organismos descomponedores del suelo (bacterias y hongos) a materiales libres de carbono (inorgánicos).

4. Para cada **provincia** la aplicación guardará su código del INE y su nombre. El Instituto Nacional de Estadística (INE) le asocia a cada provincia española un código. Este código también es empleado en la agricultura y en el catastro para identificar a las provincias.

5. Un agricultor podrá agrupar varias fincas en una **explotación** o crear una explotación con una única finca. Para ello, aportará un nombre a la explotación. Una explotación solamente es una unidad conceptual por la cual se agrupan varias fincas.

6. La aplicación almacenará para cada finca un tipo de **cultivo**. La cantidad de cultivo que produce una finca, el rendimiento, se medirá en kilogramos por hectárea (kg/ha).

    - La aplicación ya tendrá ciertos tipos de cultivo previamente definidos de los cuales almacena: el nombre, el nombre científico, la familia y el rendimiento. La aplicación también debe almacenar las extracciones minerales que cada cultivo realiza del suelo. Es decir, el nitrógeno, fósforo, potasio, azufre, calcio, magnesio, manganeso, boro, cobre y zinc que el cultivo toma del suelo para su desarrollo. Estas extracciones minerales son las se deben reponer mediante abonos. Las extracciones se medirán en kilogramos de mineral extraído por tonelada de cultivo (kg/t), como una tonelada son mil kilos es un tanto por mil. Igualmente, la aplicación almacenará si el cultivo está en simbiosis, es decir, si las bacterias en las raíces del cultivo aportan beneficios al cultivo y el cultivo a las bacterias. La aplicación también almacenará la profundidad de las raíces del cultivo (m) y la limitación anual de nitrógeno por cultivo, hectárea y año.
    - Además, el agricultor podrá añadir nuevos cultivos a la aplicación, pero no tiene por qué conocer las extracciones minerales que realiza ese cultivo que él quiere añadir. Entonces la aplicación calculará las extracciones de minerales asociadas a ese cultivo, pero para ello el agricultor tiene que introducir los porcentajes de humedad del cultivo, de proteína y de contenido de nitrógeno. La aplicación calcula las extracciones minerales, es decir, las extracciones de nitrógeno, fósforo, potasio…

7. El agricultor podrá insertar en la aplicación los **restos de cosecha**. Es decir, las toneladas por hectárea (t/ha) de cosecha que no ha recogido. Esto es importante ya que los restos de cosecha añaden kilos de nitrógeno por tonelada de residuo. La aplicación tendrá almacenada para cada cultivo la riqueza de nitrógenos de los restos.

8. La aplicación asociará un **fraccionamiento de abonado** distinto para cada cultivo. Es decir, unos porcentajes orientativos, en los que el agricultor dosificará el abono que necesita el cultivo. Se realiza así ya que una sola aplicación no se ajustaría correctamente al estado de desarrollo del cultivo. La aplicación deberá almacenar el número de aplicación (primera, segunda...), el porcentaje del abono que se debe añadir en esa aplicación y el momento óptimo para aplicarlo a la finca.

9. Los agricultores podrán añadir a una finca uno o varios **abonos orgánicos**. Por lo tanto, en la aplicación una finca tendrá asociados cero o varios abonos orgánicos. La cantidad de abono orgánico en una finca se medirá en kilogramos por hectárea. El agricultor indicará a la aplicación si ha añadido esos abonos en fondo y/o en cobertera. Es decir, si los ha añadido antes de plantar el cultivo o después.

    La aplicación ya tendrá predefinidos ciertos tipos de abonos orgánicos. Para cada abono orgánico predefinido la aplicación almacena: la especie animal de la que procede, el tipo de residuo y la procedencia residuo (fosa, balsa, estercolero…). Para algunos también guarda la actividad de la granja (cebadero, madres, leche…), el alojamiento, el tipo bebedero y las características del residuo. Igualmente, la aplicación guardará la riqueza

del abono en nitrógeno total, nitrógeno amoniacal, fósforo y potasio medido en kg de mineral por 100 kg de abono, es un %. También almacenará la riqueza de calcio, magnesio, sodio, cobre y zinc medida en tanto por ciento. También guardará el porcentaje de materia seca y de materia orgánica.

El agricultor podrá añadir nuevos tipos de abonos orgánicos indicando obligatoriamente el nombre, la especie, el tipo, el nitrógeno total, el fósforo y el potasio.

10. Si un año se añade abono orgánico a una finca, los dos años siguientes un porcentaje de este seguirá haciendo efecto en la finca. Esto es lo que se denomina la **eficiencia del abono orgánico**. Esta debe indicar para un abono y un cultivo en concreto el porcentaje del abono que hace efecto al año siguiente y el porcentaje que hace efecto el segundo año. Es decir que si este año un agricultor añade un abono orgánico a una finca, al año siguiente igual un 30% de los minerales de ese abono siguen haciendo efecto en ese finca y al cabo de dos años igual es sólo un 10%.

11. Los agricultores podrán añadir a una finca uno o varios **abonos minerales**. Por lo tanto, en la aplicación una finca tendrá asociados cero o varios abonos minerales. La cantidad de abonos minerales se medirá en kilos por hectárea. El agricultor indicará a la aplicación si se aplicarán en fondo o en cobertera.

    - La aplicación ya tendrá predefinidos ciertos tipos de abonos minerales. Para cada abono mineral predefinido la aplicación almacenará el nombre, fabricante, tipo y subtipo. También guardará si se puede aplicar en fondo y/o si se puede aplicar en cobertera, la forma del nitrógeno, para el tipo de suelo y cultivo que es idóneo, la forma de aplicación y el porcentaje de asimilación.
    - Un abono mineral sólo se podrá usar en ciertos cultivos, no en todos. Por lo tanto la aplicación almacenará en qué cultivos se puede usar un abono mineral.
    - Además, la aplicación guardará el proceso de transformación, la velocidad de liberación y los cultivos sobre los que se aplica.
    - La aplicación también almacenará el tanto por ciento (kg de mineral/100 kg de abono) de riqueza mineral en nitrógeno total, nitrógeno amoniacal, nitrógeno nítrico, nitrógeno ureico, fósforo, potasio, azufre, magnesio, hierro, boro y zinc.
    - Por otro lado, guardará la solubilidad en agua, es decir los gramos por litro sin que haya precipitación (es decir los gramos que es capaz de asimilar).
    - El agricultor podrá añadir nuevos tipos de abonos minerales indicando obligatoriamente el nombre, si se puede aplicar en fondo y si se puede aplicar en cobertera. También deberá añadir como mínimo uno de los aportes de nitrógeno, fósforo o potasio.

REQUISITOS NO FUNCIONALES

1. El cliente ha informado de que lo idóneo sería crear una aplicación web y una aplicación móvil. Sin embargo tras señalarle que debido a la limitación de tiempo del TFG quizá no fuera posible realizar las dos, ha decidido darle prioridad a la aplicación web.

2. En la aplicación debe haber unos datos previamente introducidos, que el cliente me va a facilitar. Estos son:
   a) Unos 100 tipos diferentes de cultivos junto a sus características.
   b) Unos 15 tipos de abonos minerales.
   c) Unos 30 tipos de abonos orgánicos.
   d) Los fraccionamientos de abonado de ciertos cultivos, para informar al agricultor cómo debe aplicar los abonos para ese cultivo.
   e) Las eficiencias de los diferentes abonos orgánicos para los diferentes cultivos. Es decir, para cada abono orgánico y cada cultivo el porcentaje de abono que afecta los dos años siguientes.
   f) La riqueza de nitrógeno de los restos de cada uno de los cultivos.

3. Esta aplicación será usada por agricultores, que no tienen por qué estar muy acostumbrados a emplear aplicaciones web. Por lo tanto, la interfaz debe ser sencilla de utilizar e intuitiva.

4. El cliente necesita que la aplicación esté en funcionamiento para el mes de julio ya que desea emplearla para los cálculos de su TFG de Ingeniería Agrícola. Su TFG lo entregará en septiembre, ya que su carrera tiene unos plazos diferentes a los del resto de la facultad.

5. La aplicación necesitará una importante carga de datos inicial. Para ello el cliente me pasará los datos en formato Excel y yo los tendré que añadir a la base de datos.

6. El cliente explicará y entregará las diferentes fórmulas que el programa necesita para calcular los abonados.

ALCANCE

Como he citado en los requisitos no funcionales el cliente se compromete a aportar los datos iniciales y también las fórmulas necesarias para realizar los cálculos que se implementarán en la aplicación.

Los requisitos funcionales que he presentado contienen todas aquellas funcionalidades con las que al cliente le gustaría contar en la aplicación. En ellos he reflejado todo lo que el cliente me ha dicho que era interesante que hiciera la aplicación, pero no he tenido en cuenta la limitación horaria propia del TFG, de 300 horas. Por lo tanto, el alcance inicial del trabajo se limitará a la implementación de la estructura de datos necesaria para poder desarrollar sobre ella las funcionalidades presentadas en los requisitos funcionales, así como a la implementación de las fórmulas básicas de cálculo de los abonos necesarios. Considero que la enumeración completa de los requisitos funcionales y no funcionales es de utilidad en sí misma por si consigo implementar la parte básica del sistema en un tiempo menor del previsto o por si en el futuro se decide continuar con la realización de la aplicación.

Aparte de la memoria, no me comprometo a crear una documentación adicional, ya que la interfaz de uso deberá ser suficientemente intuitiva y la memoria deberá servir también como documentación de cómo se ha implementado la aplicación. Una vez entregada la aplicación al cliente, y recibido su visto bueno, no me encargaré de su futuro mantenimiento.

ESTUDIO DE POSIBLES SOLUCIONES

Para resolver este problema lo primero que hice es un estudio de mercado para ver si ya existía alguna aplicación que cumplía los requisitos del cliente. Tras buscar en la web sí que descubrí que existían aplicaciones similares, pero o no eran libres o no cumplían los requisitos.

Una aplicación que encontré es MesParcelles [1], pero aparte de estar en francés, es de pago. Otra aplicación que localicé es la calculadora de fertilizantes de la marca Fertiberia [2], pero aparte de sólo tener los fertilizantes de su marca, no permite elegir fertilizantes orgánicos. También vi la calculadora de fertilizantes de smart-fertilizer [3] pero también era de pago, el software más barato que venden son 300$.

Teniendo en cuenta lo anterior determiné que la mejor opción era crear una aplicación web propia.

METODOLOGÍA DE TRABAJO

Como he descrito en el punto anterior voy a tener que desarrollar una aplicación desde cero. Para ello primero voy a emplear una metodología en cascada para programar la funcionalidad básica de la aplicación. En el caso de que dé tiempo a añadir alguna otra funcionalidad emplearemos una metodología iterativa.

Para la funcionalidad básica, siguiendo el diseño en cascada, primero realizaremos un análisis. El análisis primero contendrá el estudio de requisitos funcionales y no funcionales con las características que desea el cliente. A continuación, haremos un análisis de los casos de uso que podrán realizar los usuarios de la aplicación. También analizaremos los cálculos que debe realizar la aplicación. Por último, haremos un estudio de las mejores tecnologías de servicio web para aplicar en ese caso.

Como la metodología en cascada indica la siguiente fase es el diseño. En ella lo primero que voy a realizar es un diseño de la base de datos. También realizaré un diseño del programa, es decir, de las diferentes clases que voy a tener que emplear y los métodos y funciones de cada una de ella.

La siguiente fase que voy a realizar es la implementación del programa en las tecnologías elegidas. Esta fase se va a dividir en cuatro subfases. La primera va a ser la instalación de las tecnologías necesarias para la implementación. La segunda va a ser la creación de las bases de datos. La tercera va a ser la programación del servicio web y la cuarta la creación de la interfaz gráfica. En estas dos últimas fases me voy a desviar del desarrollo en cascada típico ya que según vaya programando diferentes funcionalidades se las voy a ir enseñando al cliente. De esta forma, el cliente va probando poco a poco las diferentes funcionalidades y me va indicando qué aspectos deben ser mejorados.

Una vez realizada la funcionalidad básica, si sobra tiempo, añadiré cierta funcionalidad extra que pueda desear el cliente.

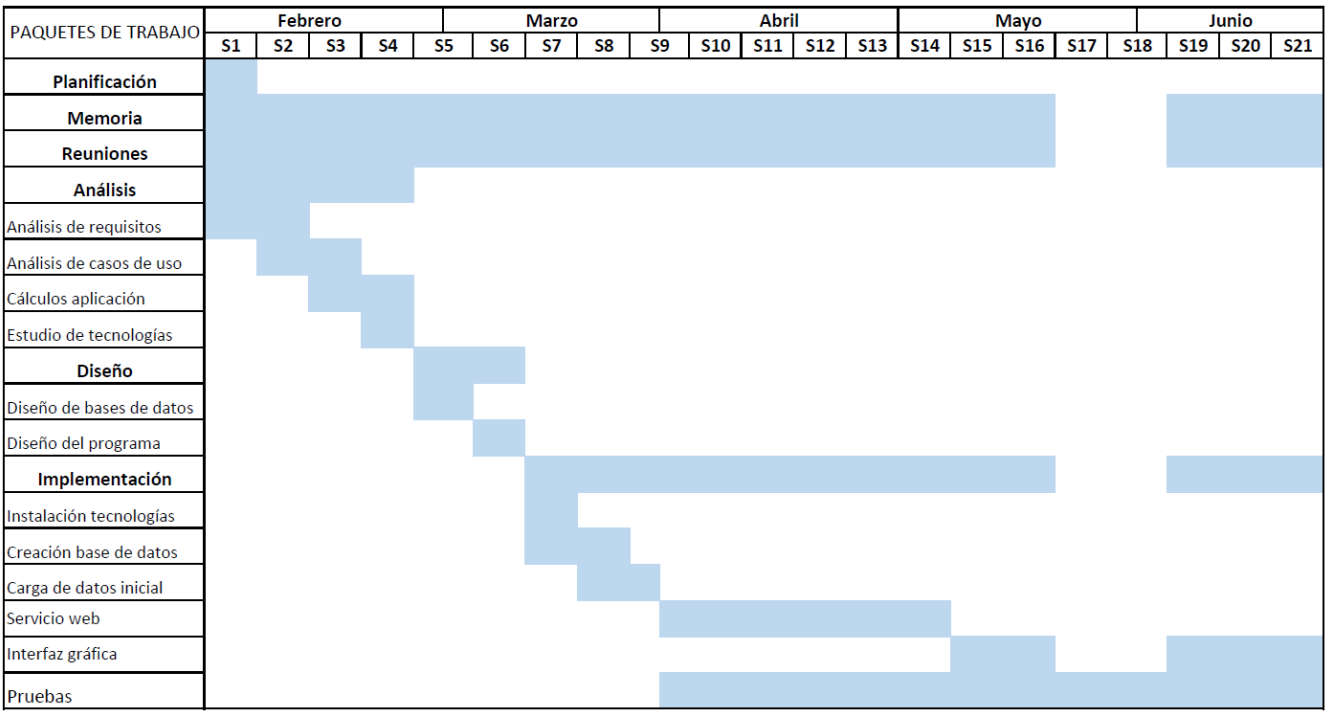
[1] http://www.mesparcelles.fr/
[2] http://www.fertiberia.com/es/agricultura/servicios-al-agricultor/calculadora/
[3] http://www.smart-fertilizer.com/es/articles/Fertilizer-Calculator

A continuación, añado la dedicación en horas que le voy a dedicar a cada tarea:

TAREA	HORAS
Memoria + Reuniones + Gestión	78
Análisis de requisitos	12
Análisis de casos de uso	12
Cálculos aplicación	12
Estudio de tecnologías	6
Diseño de bases de datos	12
Diseño del programa	12
Instalación tecnologías	3
Creación base de datos	15
Carga de datos inicial	12
Servicio web	66
Interfaz gráfica	60
TOTAL	300

DIAGRAMA DE GANTT

En la siguiente figura podemos observar un diagrama de Gantt en el cual describo cuándo voy a realizar cada una de las tareas.



ANÁLISIS DE LA APLICACIÓN

ANÁLISIS DE CASOS DE USO

A continuación, voy a explicar en detalle el caso de uso “Obtener cantidad de abono necesario para una finca” ya que es la funcionalidad principal de la aplicación. El resto de casos de uso no los detallaremos porque su complejidad no lo requiere.

Caso de uso: Obtener cantidad de abono necesario para una finca
Objetivo: Informar al agricultor de cuánto abono debe añadir.
Actores: Usuario Registrado (U), Sistema(S).
Precondición: Que el usuario esté registrado, que la finca tenga un cultivo asociado
Postcondición :
Pasos: • S: El sistema calcula la cantidad de abono que el usuario necesita añadir • U: El usuario introduce la cantidad y el tipo de abono que ha añadido de fondo. Es decir, la cantidad de abono que ha puesto antes de plantar el cultivo. También el rendimiento que desea para su cultivo, los kg de cosecha por hectárea. • S: El sistema calcula la cantidad de abono que el usuario debe añadir de cobertera. Es decir, después de plantar el cultivo. Le pregunta al usuario si desea añadir abono orgánico o mineral • U: El usuario elige qué tipo de abono quiere añadir. • S: El sistema le muestra una lista con los abonos de ese tipo. • U: El usuario selecciona el abono que desea. • S: El sistema le indica la cantidad de abono para añadir en cobertera.
Extensiones:

CÁLCULOS QUE REALIZA LA APLICACIÓN

Cálculo de cantidad de abono a aportar

1. Primero el usuario debe informar del rendimiento esperado para el cultivo, es decir la cantidad de kilogramos de cultivo que se pretenden obtener por hectárea (kg/ha).
2. A continuación, la aplicación debe calcular qué cantidad de los diferentes minerales debe añadir el agricultor al suelo para compensar las extracciones. En este caso los minerales que vamos a tener en cuenta son nitrógeno (N), fósforo (P) y potasio (K).

Por un lado, para calcular las extracciones del cultivo, se multiplica el rendimiento esperado por hectárea (kg/ha) por la extracción de mineral que realiza el cultivo (kg/1000 kg). Este último dato ya lo tiene la aplicación almacenado para los principales cultivos. Por lo tanto, la fórmula sería:

$$\frac{kg\ de\ cosecha}{ha\ de\ superficie} * \frac{kg\ de\ mineral\ extraído}{1000\ kg\ de\ cosecha} = \frac{kg\ de\ mineral\ extraído}{1000 * ha\ de\ superficie}$$

De este modo obtenemos los kg de mineral extraído por superficie.

3. Seguidamente, la aplicación le va a preguntar al agricultor la cantidad de abono que ha añadido de fondo. Es decir, la cantidad de abono que ha añadido antes de plantar el cultivo. Esta cantidad se mide en kilogramos por hectárea de superficie (kg/ha). Para ello el agricultor seleccionará de una lista de abonos el tipo que ha añadido. Todos los abonos tienen asociada la cantidad de mineral que aportan, medido en kg de mineral por 100 kilos de abono (%). De esta forma obtenemos la cantidad de kilos de mineral por hectárea que ha añadido con el abonado de fondo (kg/ha) (La fórmula sería similar a la del punto 2, pero dividido entre 100).

4. Para calcular la cantidad que necesita de un mineral determinado se restará a los kg/ha de mineral extraídos por el cultivo (punto 2) los kg/ha de mineral que han sido aportados en el abonado de fondo (punto 3).

5. En el caso del nitrógeno también debemos sumar la cantidad de nitrógeno que ya tiene la materia orgánica del suelo. Para ello multiplicaremos la equivalencia entre metros y hectáreas (10.000 m^2 es una hectárea), por la profundidad de las raíces del cultivo (medida en cm), por la densidad del terreno (medida en g/dm^3), por la tasa de mineralización de la finca (constante), por el tanto por ciento de materia orgánica del terreno y por el contenido de nitrógeno (N) del cultivo, respecto al total de materia orgánica (%):

$$10^4 \frac{m^2}{ha} * densidad\left(\frac{kg}{m^3}\right) * profundidad\ raíz(m) * tasa\ de\ mineralización(const) * materia\ orgánica\ (\%) * nitrógeno(\%) = 10^4 \frac{kg}{ha}$$

Por lo tanto, obtenemos los kilogramos de nitrógeno por hectárea que aporta la materia orgánica del suelo y se lo restaremos al resultado del punto 4.
De este modo obtenemos los kg de mineral por hectárea que necesita la finca después del abonado de fondo.

A continuación, con los abonos de cobertera supliremos las necesidades de nitrógeno, fósforo y potasio.

6. Después la aplicación le preguntará al agricultor si el abono de cobertera lo quiere realizar con abonos orgánicos o minerales. Como ya he explicado, el abono de cobertera es el que se realiza después de haber plantado el cultivo. Entonces, le aparecerá una lista de abonos orgánicos o minerales y seleccionará el deseado. Todos los abonos tienen asociada la cantidad de nitrógeno que aportan, medido en kg de nitrógeno por 100 kilos de abono (%). Entonces para obtener la cantidad de kilos de abono por hectárea que debe añadir, se debe dividir la cantidad de kg de nitrógeno por hectárea (kg/ha) que debe añadir (punto 5), entre el nitrógeno asociado al abono de cobertera, medido en kg de mineral en 100kg de abono.

$$cantidad\ de\ abono\ añadir\ \frac{100 * kg}{ha} = \frac{cantidad\ de\ nitrógeno\ a\ aportar\ \frac{kg}{ha}}{nitrógeno\ asociado\ al\ abono\ \frac{kg}{100\ kg}}$$

Así obtenemos la cantidad de abonos a añadir en cobertera, medida en kilogramos por hectárea.

De esta manera hemos obtenido los kilos de abono de cobertera que hay que añadir para suplir las extracciones de nitrógeno del cultivo. Pero falta por estudiar si hemos suplido las necesidades de fósforo (P) y de potasio (K). Para el fósforo haremos lo siguiente:

7. A continuación, multiplico los kilogramos por hectárea (kg/ha) de abono de cobertera, que ha añadido el agricultor, por la riqueza de fósforo del abono medida en kilogramos de fósforo por 100 kg de abono (%). De esta forma obtengo los kilogramos de fósforo por hectárea que ha añadido en el abonado de cobertera (kg/ha). A esto se le sumará la cantidad de fósforo que añadió en el abonado de fondo (kg/ha) (punto 3).

8. Después a la cantidad de fósforo añadido (punto 7) le restamos la cantidad de fósforo que necesitaba (punto 4). Normalmente se obtendrá que se ha añadido fósforo en exceso. En el caso de que se necesite más fósforo se le permitirá elegir entre un número de abonos minerales de cobertera que no tengan nitrógeno. Realizando los mismos cálculos que para el nitrógeno la aplicación calculará la cantidad de abonos que debe añadir (punto 6).

Finalmente para el potasio, la aplicación realizará los mismos cálculos que ha realizado para el fósforo, pero si hemos añadido más abono de cobertera para suplir la falta de fósforo, se tendrá en cuenta. En la mayor parte de casos no se necesitará más potasio. En el caso de que sí se necesite más, se permitirá elegir al cliente un abono mineral que sólo tenga potasio y la aplicación le indicará la cantidad que debe añadir.

Para la aplicación el cliente ha decidido emplear estas fórmulas simplificadas para obtener la cantidad de abono de cobertera que se debe añadir a una finca. Digo que son simplificadas porque en el requisito 7 se nombra que los restos de cosecha del año anterior aportan nitrógeno al terreno, por lo tanto influyen ligeramente en la cantidad de abono que hay que añadir a la finca. Sin embargo en estas fórmulas no se ha tenido en cuenta.

Además, en las fórmulas del cliente tampoco se ve reflejado el requisito 10 sobre la eficiencia de un abono orgánico. Este requisito dice que los dos años siguientes al que ha sido plantado el abono orgánico, un pequeño porcentaje de este sigue haciendo efecto.

Aun así los datos tomados en las fórmulas simplificadas tienen mucha más influencia en la cantidad de abono que hay que añadir que los datos que no se han tenido en cuenta.

ANÁLISIS TECNOLÓGICO

En esta aplicación he decidido que voy a emplear el **Modelo Vista Controlador (MVC)**. Este modelo tiene tres capas bien diferenciadas:

El **modelo** es la capa de persistencia, es decir, de acceso a la base de datos. Y también es la capa de lógica de negocio.

El **controlador** se encarga de recibir las peticiones (del usuario) desde las vistas, controla los accesos, invoca a los servicios necesarios del modelo, obtiene el resultado, selecciona la vista y le entrega el resultado.

La **vista** se muestra, usando datos del modelo entregados por el controlador. Incluye enlaces-formularios-botones para las nuevas peticiones del usuario.

Esto permite que las vistas no incluyan nada de la lógica de la aplicación y el mismo controlador pueda servir para varias vistas. Lo cual puede ser interesante si en el futuro a la interfaz se le quiere dar un aspecto más profesional o cambiarla completamente. De este modo sólo tendrían que modificar la vista y podrían trabajar con el mismo modelo y con el mismo controlador.

Además como cada parte se puede modificar sin afectar a las demás, si en un futuro se quiere cambiar funcionalidades, como por ejemplo calcular la cantidad de abono con una técnicas más complejas, sólo tendrían que cambiar la lógica de negocio del modelo.

También tiene la ventaja de que el código está estructurado, por lo tanto si alguien decide modificar esta aplicación en el futuro le va a resultar mucho más sencillo.

Por otro lado tiene ventajas adicionales como que si se produce un error en la aplicación, se puede detectar antes de haber empezado a devolver datos a la vista.

Para el acceso a la base de datos vamos a emplear un ORM ya que:

- **Mejora la productividad**. Elimina la tarea de codificación de las consultas SQL de las tablas de las BD.
- **Facilita el mantenimiento**. Al tener menos líneas de código, el sistema es más comprensible y fácil de mantener, ya que se centra en la lógica de negocio mientras el ORM se encarga de la transformación entre la BD- aplicación.
- I**ndependencia y abstracción del SGBD**. Abstraen la aplicación del lenguaje SQL de la BD y del dialecto SQL. Si el framework ORM soporta múltiples BD, la aplicación además de ser independiente de la BD, es portable entre las distintas BD.

Como voy a programar con Java el ORM que voy a emplear es JPA. El API Java Persistence API (JPA) proporciona un modelo de mapeo O/R para gestionar los datos relacionales en aplicaciones Java, es decir, para **acceder**, **persistir** y **gestionar** datos entre clases Java y una BD relacional. Voy a emplear JPA porque en la asignatura de programación de bases de datos vimos que se suele emplear este lenguaje.

CAPÍTULO 2. DISEÑO

En este capítulo se va a realizar un diseño de la base de datos a partir de los datos de los requisitos. Para ello voy a realizar un diagrama UML, el cual posteriormente voy a transformar en tablas. Además, voy a explicar las relaciones entre las distintas tablas.

DISEÑO DE LA BASE DE DATOS

La primera tabla que pensé que había que añadir es la **tabla agricultor**. En esa tabla decidí que el nombre del usuario debería ser la clave primaria de esta tabla, ya que como a la hora de logearse sólo introducen el nombre de usuario y la contraseña, deberán ser identificados por el nombre de usuario.

También decidí crear una **tabla explotación**. Como dos agricultores pueden querer poner el mismo nombre a su explotación decidí que la clave primaria de la explotación debía ser un valor que se autoincrementase. Una explotación tendrá una o varias fincas. Además una finca estará asociada al agricultor que la ha añadido.

En la **tabla finca**, por la misma razón que en la tabla explotación, decidí que la clave primaria de las fincas fuese un valor autoincremental. A la finca se le asociará el agricultor que la ha introducido en la base de datos. Una finca también estará asociada a una explotación. Además, la finca estará asociada a una provincia. En la **tabla provincia** el id será el número de la provincia según el Instituto Nacional de Estadística. Además, una finca tendrá asociado un cultivo.

La clave primaria de la **tabla cultivo** será también un valor auto-incremental por la misma razón que la explotación y con la finca. Un cultivo si está en la base de datos no tiene que tener ningún agricultor asociado, pero si se añade uno nuevo estará asociado al agricultor que lo añade. Un cultivo puede estar asociado con varias fincas o con ninguna. Un cultivo tendrá unos abonos orgánicos asociados.

La base de datos también tendrá una **tabla abono orgánico** cuya clave principal será un valor autoincremental. Un abono orgánico si está en la base de datos no tiene que tener ningún agricultor asociado, pero si se añade uno nuevo estará asociado al agricultor que lo añade.

Además, un abono orgánico podrá estar asociado a una finca, en su relación se deberá guardar la cantidad de abono orgánico que se ha añadido y si se ha añadido en fondo o en cobertera. Puede ser que se utilice un mismo abono orgánico en una misma finca tanto para el abonado de fondo como de cobertera. Por lo tanto ,en la **tabla** que surge en la relación entre abono orgánico y finca la clave primaria estará formada por el id del abono, el id de la finca y un booleano que indica si se ha añadido de fondo o de cobertera.

La base de datos también tendrá una **tabla abono mineral** cuya clave principal será un valor autoincremental. Un abono mineral si está en la base de datos no tiene que tener ningún agricultor asociado, pero si se añade uno nuevo estará asociado al agricultor que lo añade.

Además, un abono mineral podrá estar asociado a una finca, en su relación se deberá guardar la cantidad de abono orgánico que se ha añadido y si se ha añadido en fondo o en cobertera. Puede ser que se utilice un mismo abono orgánico en una misma finca tanto para el abonado de fondo como de cobertera. Por lo tanto, en la **tabla** que surge en la relación entre abono orgánico y finca la clave primaria estará formada por el id del abono, el id de la finca y un booleano que indica si se ha añadido de fondo o de cobertera.

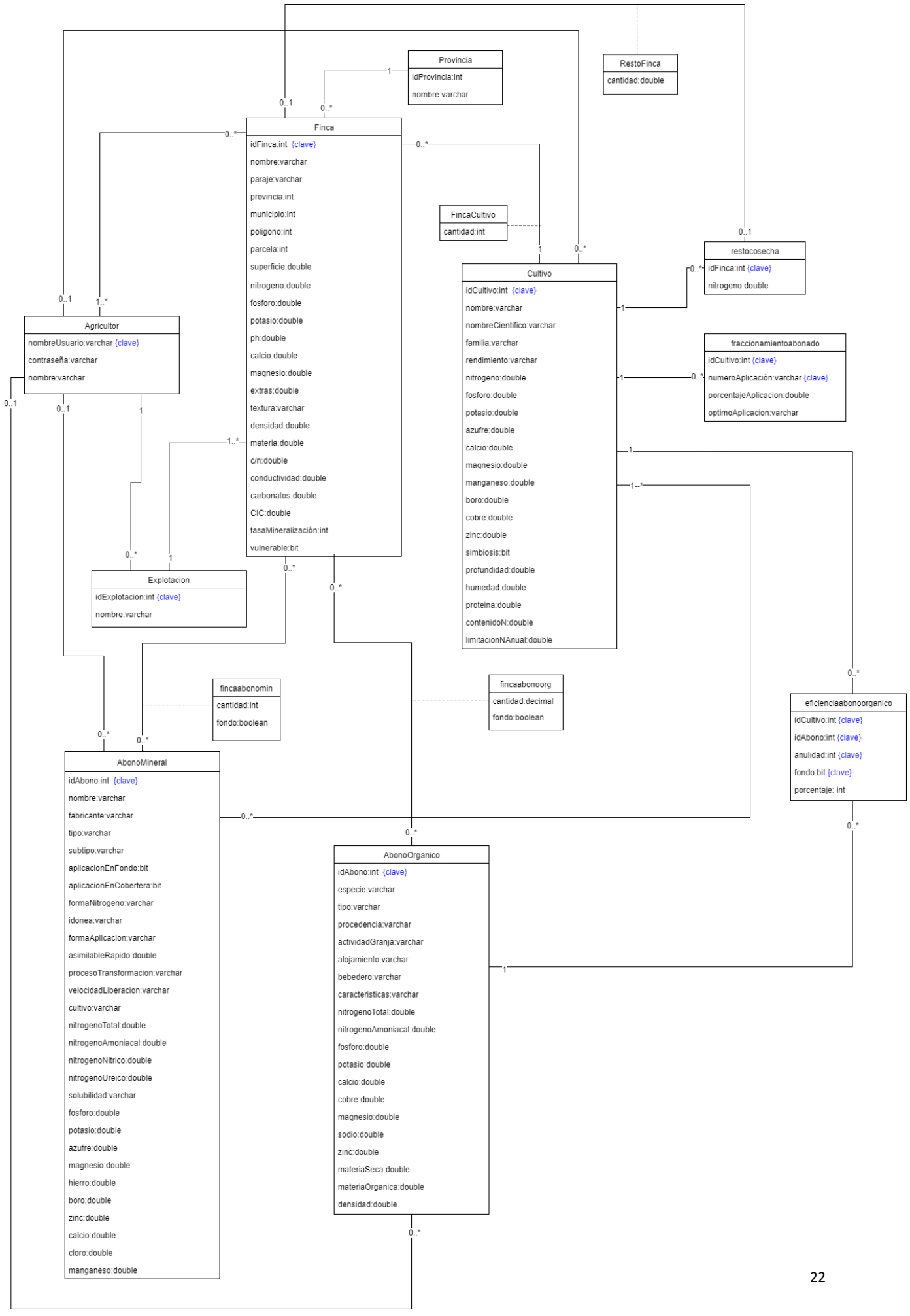
Por otro lado, no todos los abonos minerales sirven para todos los cultivos, por lo tanto habrá una relación en la cual un abono mineral estará como mínimo asociado a un cultivo y un cultivo podrá tener más de un abono mineral asociado.

También hice una **tabla para la eficiencia del abono orgánico**. Esta tiene como clave primaria el id del abono, el id del cultivo, la anualidad y si el abono era de fondo o de cobertera. Además, la tabla tendrá otra columna con el porcentaje que hace efecto en esa anualidad. Una eficiencia estará asociada a un cultivo y a un abono. Sin embargo, un cultivo puede tener varias eficiencias y un abono también puede tener varias eficiencias. He tenido que poner como clave primaria tantos campos porque el porcentaje que hace efecto depende del abono, el cultivo, si el abono se emplea de fondo o cobertera y de hace cuántos años se plantó el cultivo (anualidad).

Me di cuenta que necesitaba una **tabla para el fraccionamiento del abonado** en un cultivo. La clave primaria de esta tabla es el id del cultivo y el número de aplicación. También tendrá una columna que indique el porcentaje de abono que hay que añadir en esa aplicación e información sobre cuándo es óptimo añadirla. El fraccionamiento de abonado estará asociado a un cultivo y un cultivo tendrá varios fraccionamientos de abonado.

Creé una **tabla restos de cosecha**. Esta tabla va a tener como clave primaria el identificador de la finca. Igualmente va a guardar el nitrógeno de los restos de cosecha del cultivo. Una finca va a tener un resto de cosecha (el del año pasado). En la relación entre finca y restos de cosecha se guarda la cantidad de cultivo que no se ha recogido. La tabla tendrá una columna que hará referencia al cultivo del cual son los restos. Y otra columna con la cantidad de N que aportan los restos.

Esta última tabla desde el punto de vista de diseño de bases de datos no haría falta, porque se podría hacer como una relación entre finca y cultivo sin tabla intermedia. Sin embargo pensándolo desde el punto de vista de la programación los restos de una cosecha es como un abono ya que aporta nutrientes al suelo. Por eso he decidido dejarlo como una tabla aparte ya que a la hora de programar podría heredar de una superclase abono.


Provincia
idProvincia:int
nombre:varchar
RestoFinca
cantidad:double
Finca
idFinca:int {clave}
nombre:varchar
paraje:varchar
provincia:int
municipio:int
poligono:int
parcela:int
superficie:double
nitrogeno:double
fosforo:double
potasio:double
ph:double
calcio:double
magnesio:double
extras:double
textura:varchar
densidad:double
materia:double
c/n:double
conductividad:double
carbonatos:double
CIC:double
tasaMineralización:int
vulnerable:bit
FincaCultivo
cantidad:int
Cultivo
idCultivo:int {clave}
nombre:varchar
nombreCientifico:varchar
familia:varchar
rendimiento:varchar
nitrogeno:double
fosforo:double
potasio:double
azufre:double
calcio:double
magnesio:double
manganeso:double
boro:double
cobre:double
zinc:double
simbiosis:bit
profundidad:double
humedad:double
proteina:double
contenidoN:double
limitacionNAnual:double
restocosecha
idFinca:int {clave}
nitrogeno:double
fraccionamientoabonado
idCultivo:int {clave}
numeroAplicación:varchar {clave}
porcentajeAplicacion:double
optimoAplicacion:varchar
Agricultor
nombreUsuario:varchar {clave}
contraseña:varchar
nombre:varchar
Explotacion
idExplotacion:int {clave}
nombre:varchar
fincaabonomin
cantidad:int
fondo:boolean
fincaabonoorg
cantidad:decimal
fondo:boolean
eficienciaabonoorganico
idCultivo:int {clave}
idAbono:int {clave}
anulidad:int {clave}
fondo:bit {clave}
porcentaje: int
AbonoMineral
idAbono:int {clave}
nombre:varchar
fabricante:varchar
tipo:varchar
subtipo:varchar
aplicacionEnFondo:bit
aplicacionEnCobertera:bit
formaNitrogeno:varchar
idonea:varchar
formaAplicacion:varchar
asimilableRapido:double
procesoTransformacion:varchar
velocidadLiberacion:varchar
cultivo:varchar
nitrogenoTotal:double
nitrogenoAmoniacal:double
nitrogenoNitrico:double
nitrogenoUreico:double
solubilidad:varchar
fosforo:double
potasio:double
azufre:double
magnesio:double
hierro:double
boro:double
zinc:double
calcio:double
cloro:double
manganeso:double
AbonoOrganico
idAbono:int {clave}
especie:varchar
tipo:varchar
procedencia:varchar
actividadGranja:varchar
alojamiento:varchar
bebedero:varchar
caracteristicas:varchar
nitrogenoTotal:double
nitrogenoAmoniacal:double
fosforo:double
potasio:double
calcio:double
cobre:double
magnesio:double
sodio:double
zinc:double
materiaSeca:double
materiaOrganica:double
densidad:double

TABLAS DE LA BASE DE DATOS

AGRICULTOR		
nombreUsuario	contrasenia	nombre

FINCA								
idFinca	nombre	paraje	superficie	nitrogeno	potasio	fosforo	extras	provincia

CE:provincia

municipio	poligono	parcela	textura	densidad	ph	calcio	magnesio

materia	C/N	conductividad	carbonatos	CIC	vulnerable	tasaMineralización

explotacion	cultivo	cantidad

CE:explotación CE:cultivo

FINCAAGRICULTOR	
finca	agricultor

CE:finca CE:agricultor

PROVINCIA	
idProvincia	nombre

EXPLOTACION		
idExplotacion	nombre	agricultor

CE:agricultor

CULTIVO					
idCultivo	nombre	nombreCientifico	familia	rendimiento	nitrogeno

fosforo	potasio	azufre	calcio	magnesio	manganeso	boro	cobre	zinc

simbiosis	profundidad	humedad	proteina	contenidoN	limitacionNAnual	agricultor

CE:agricultor

RESTOCOSECHA			
idFinca	cultivo	nitrogeno	cantidad

CE:finca CE:cultivo

FRACCIONAMIENTOABONADO			
idCultivo	numeroAplicacion	porcentajeAplicacion	optimoAplicacion

CE:cultivo

ABONOORGANICO						
idabonoorganico	especie	tipo	procedencia	actividadGranja	alojamiento	bebedero

caracteristicas	nitrogenoTotal	nitrogenoAmoniacal	fosforo	potasio	calcio	cobre

magnesio	sodio	zinc	materiaSeca	materiaOrganica	densidad	agricultor

CE:agricultor

FINCAABONOORG			
finca	abonoOrg	fondo	cantidad

CE:finca CE:abonoorganico

EFICIENCIAABONOORGANICO				
abonoorganico	cultivo	anualidad	fondo	porcentaje

CE:abonoOrganico CE:cultivo

ABONOMINERAL						
idabonomineral	nombre	fabricante	tipo	subtipo	aplicacionEnFondo	aplicacionEnCobertera

formaNitrogeno	idonea	formaAplicacion	asimilableRapido	procesoTransformacion

velocidadLiberacion	cultivo	nitrogenoTotal	nitrogenoAmoniacal	nitrogenoNitrico

nitrogenoUreico	solubilidad	fosforo	potasio	azufre	magnesio	hierro	boro

zinc	calcio	cloro	manganeso	agricultor

CE:agricultor

FINCAABONOMIN			
abonoMin	finca	fondo	cantidad

CE:abonomineral CE:finca

CULTIVOABONOMIN	
abonoMin	cultivo

CE:abonomineral CE:cultivo

ONUPDATE Y ONDELETE EN LAS TABLAS

A la hora de crear una base de datos es muy importante definir las acciones que se van a realizar cuando se intenta eliminar o actualizar una clave a la que apuntan las claves externas existentes. Para ello existen respectivamente las operaciones on delete y on update, las cuales se pueden definir con diferentes valores. En esta base de datos he empleado el valor **cascade**, el cual especifica que si se intenta eliminar o modificar una fila con una clave a la que hacen referencia claves externas de filas existentes en otras tablas, todas las filas que contienen dichas claves externas también se eliminan o modifican. También he empleado el valor **set null** el cual especifica que si se intenta eliminar o modificar una fila con una clave a la que hacen referencia

las claves externas de las filas existentes de otras tablas, todos los valores que conforman la clave externa de las filas a las que se hace referencia se establecen en NULL.

A continuación, voy a describir cómo he definido estas operaciones para las diferentes tablas:

La tabla **provincias** no se va a poder modificar ni eliminar, por lo tanto, no he definido estas operaciones.

En la tabla **fincas**, si se modifica o borra una finca también se debían borrar las columnas de las tablas que hacen referencia a esta, las cuales eran: fincaabonoorg, fincaabonomin y fincaagricultor. Por lo tanto, las claves extranjeras a finca las he declarado como onUpdate y onDelete de tipo cascade. Ver los requisitos funcionales 3, 9 y 11.

Si un **agricultor** modifica sus datos o se elimina también habrá que modificar o eliminar los datos de las tablas que le hacen referencia, que son: abonomineral, abonoorganico, cultivo, fincagricultor y explotación. Por lo tanto, las claves extranjeras a finca las he declarado como onUpdate y onDelete de tipo cascade. Ver los requisitos funcionales 3, 6, 9 y 11.

En la tabla **cultivos**, si se modifica o borra un cultivo también se dederán modificar o borrar las columnas de ciertas tablas que hacen referencia a este, las cuales eran: restocosecha, fraccionamientoabonado y eficienciabonoorganico. Por lo tanto, las claves extranjeras a finca las he declarado como onUpdate y onDelete de tipo cascade. Sin embargo a pesar de que la tabla finca tiene una clave extranjera a cultivo no queremos que se elimine la finca porque se elimine el cultivo por lo tanto pondremos onUpdate cascade y onDelete set null. Ver los requisitos funcionales 6, 7, 8 y 11.

A la tabla **explotación** sólo le hace referencia la tabla finca. OnUpdate lo pondremos como cascade porque si se modifican los datos de la explotación también queremos que se modifique la referencia de la finca, pero onDelete lo ponemos como setNull porque igual queremos eliminar una explotación, pero asociar la finca a otra explotación. Ver el requisito funcional 5.

A la tabla **abonoorganico** le hace referencia fincaabonoorg y eficienciaabonoorganico, si un abono orgánico se modifica o se borra también se deberá modificar o borrar en estas tablas. Por ello, tanto onUptade como onDelete lo he puesto como cascade. Ver los requisitos funcionales 9 y 10.

A la tabla **abonomineral** le hace referencia fincaabonomin y cultivoabonomin, si un abono mineral se modifica o se borra también se deberá modificar o borrar de estas tablas. Por ello, tanto onUptade como onDelete lo he puesto como cascade. Ver el requisito funcional 11.

REALIZACIÓN DE LA BASE DE DATOS

Para realizar la base de datos me planteé dos programas: SQL Server y MySQL, pero tras estudiar las características de ambos me decanté por el segundo debido a las siguientes razones:

- MySQL es multiplataforma, es decir, es compatible con Windows, Linux y Mac. Sin embargo, SQL Server solamente es compatible con Windows.
- La versión gratuita de SQL Server, denominada Express, tiene un límite de tamaño de la base de datos de 10 GB. El tamaño de la base de datos en MySQL sólo está limitado por el sistema operativo.
- MySQL es una herramienta que se suele utilizar más para operaciones relacionadas con la gestión de bases de datos asociada a un sitio web.

- MySQL es más sencillo de instalar que SQLServer.

Otras razones por las que me gusta MySQL es porque tiene el programa MySQL Workbench, que permite trabajar de forma muy intuitiva y sencilla. Además de que se pueden hacer consultas mediante JDBC.

También que tiene una extensión para Excel denominada MySQL for Excel que permite importar una tabla desde MySQL y modificar los datos. También permite tomar datos de una tabla Excel y exportarlos a una tabla de MySQL. Este aspecto es importante en mi aplicación ya que voy a tener que realizar una gran carga de datos iniciales y estos datos me los va a pasar el cliente en formato Excel.

Una de las decisiones en las que dudé es como poner los números reales en MySQL, ya que tiene las siguientes opciones: float, double, numeric y decimal. Finalmente me decidí por el tipo double ya que posteriormente, al hacer el mapeo mediante ORM, su tipo asociado de Java es double y este tipo es el que más se suele emplear en java para los números reales. A pesar de que el tipo decimal tenga una mayor precisión [1] en este caso la aplicación no necesita tanta precisión.

La mayor dificultad surgió a la hora de añadir los datos a las tablas, ya que el cliente me había proporcionado los datos en tablas de Excel. Para ello intenté diferentes formas de introducir los datos.

La primera fue mediante el comando de MySQL load data infile, pero por me salía un error que a pesar de que busqué posibles soluciones en stack over flow, no funcionaba correctamente.

Después instalé la extensión de MySQL para Excel. El problema es que esta tampoco me servía ya que no permite dejar campos de la tabla de bases de datos sin mapearle ninguna columna de la tabla de Excel.

Finalmente decidí crear un programa en Java para poder importar correctamente los datos desde Excel. Para ello primero guardé el Excel con formato CSV delimitado por comas. Como las tablas tienen codificación UTF8 mediante el programa Notepad cambié la codificación a UTF-8 sin boom.

Luego con el programa NetBeans y programando en Java me conecté a la base de datos empleando JDBC. Con un Buffered Reader fui leyendo las diferentes filas del csv y haciendo un insert para cada una de ellas. De esta forma, pude definir las columnas en las que quería que se insertaran datos y en las que no.

```
con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "root", "root");
if(con!=null){
    br =new BufferedReader(new FileReader("C:/Users/Alicia/Desktop/abonos.csv"));
    String line;
    while ((line=br.readLine())!=null) {
        String [] fields = line.split(";");
        System.out.println(fields.length);
        Statement stm = con.createStatement();
        stm.executeUpdate("INSERT INTO mydb.abonomineral(nombre,fabricante,tipo,subtipo,aplicacionEnFondo,aplicacionEnCobertera,"
                + "formaNitrogeno,idonea,formaAplicacion,asimilableRapido,procesoTransformacion,velocidadLiberacion,cultivo,nitrogenoTotal,"
                +"nitrogenoAmoniacal,nitrogenoNitrico,nitrogenoUreico,solubilidad,fosforo,potasio,azufre,magnesio,hierro,boro,zinc,calcio,cloro,manga
                +fields[0]+"','"+fields[1]+"','"+fields[2]+"','"+fields[3]+"',b'"+fields[4]+"',b'"+fields[5]+"','"+fields[6]+"','"
                +fields[7]+"','"+fields[8]+"','"+fields[9]+"','"+fields[10]+"','"+fields[11]+"','"+fields[12]+"','"+fields[13]+"','"
                +fields[14]+"','"+fields[15]+"','"+fields[16]+"','"+fields[17]+"','"+fields[18]+"','"+fields[19]+"','"+fields[20]+"','"
                +fields[21]+"','"+fields[22]+"','"+fields[23]+"','"+fields[24]+"','"+fields[25]+"','"+fields[26]+"','"+fields[27]+"')");
        stm.close();
    }
```


[1] https://stackoverflow.com/questions/19601975/storing-statistical-data-do-i-need-decimal-float-or-double

CAPÍTULO 3: IMPLEMENTACIÓN

Tras el diseño y la carga de datos de la base de datos comencé el mapeo ORM y la aplicación web.

Como mi aplicación iba a necesitar clases necesitaba un lenguaje que aceptase el paradigma orientado a objetos. Como Java es un lenguaje orientado a objetos y además estoy familiarizada con él, decidí usar ese lenguaje para implementar mi aplicación.

Para realizar el proyecto creé una aplicación web en NetBeans. Sin embargo, para los métodos de la capa de negocio y la capa de persistencia creé una aplicación de Java para probarlos. Posteriormente cuando estaban probados los pasaba a la aplicación web.

CAPA DE PERSISTENCIA: MAPEO ORM.

Como ya he explicado en el análisis tecnológico para trabajar desde la aplicación con la base de datos decidí realizar un mapeo ORM.

Para trabajar con ORM desde Java elegí el Java Persistence API (JPA) que proporciona un modelo de mapeo O/R para gestionar los datos relacionales en aplicaciones Java, es decir, para acceder, persistir y gestionar datos entre clases Java y una BD relacional.

En un principio creé las clases relacionadas con las tablas y luego realicé el mapeo.

Como previamente no había trabajado con ORM tuve varias dificultades y fallos que creo que es oportuno comentar.

- Mapeo de tabla intermedia con campos extra

La primera dificultad que tuve es cómo mapear una tabla intermedia con campos extra. En concreto lo necesitaba para la relación entre las tablas de finca y abono orgánico y de finca y abono mineral. Ambas relaciones son de muchos a muchos, por lo tanto necesitan una tabla intermedia cuya clave primaria está compuesta por una clave extranjera a abono, otra a finca y una columna que indica si el abono se ha añadido de fondo o de cobertera. Además tiene un campo extra para indicar la cantidad de abono que se ha añadido a una finca.

Para hacerlo busqué la solución en internet y me fijé en los siguientes enlaces de mkyong [1] y codejava [2].

Primero tuve que añadir lo siguiente a la clase finca, ya que la clase intermedia le va a apuntar con un @ManyToOne.

```
@OneToMany(fetch = FetchType.LAZY, mappedBy = "pk.finca")
private List<FincaAbonoOrg> abonosOrganicos;
```

Y en la clase del abono (mineral/organico) tuve que añadir lo siguiente, porque la clase intermedia también le va a apuntar con un @ManyToOne.

[1] https://www.mkyong.com/hibernate/hibernate-many-to-many-example-join-table-extra-column-annotation/
[2] http://www.codejava.net/frameworks/hibernate/hibernate-many-to-many-association-with-extra-columns-in-join-table-example

```
@OneToMany(fetch = FetchType.EAGER, mappedBy = "pk.abonoOrganico")
private List<FincaAbonoOrg> fincas;
```

Luego tuve que crear una clase para la tabla intermedia fincaabonoorg que apuntase a cada una de las clases correspondientes a finca y abono.

En esta clase he tenido que crear un atributo de tipo FincaAbonoOrgId, que explicaré más adelante. Este atributo es de tipo @EmbeddedId. Esta anotación es utilizada para incrustar un identificador compuesto como el identificador de esta clase. Por esto en el código superior he tenido que utilizar la notación @AssociationOverrides para sobrescribir el mapeo de los atributos finca y abonoOrganico.

Además marco los get de los atributos de la clave primaria como @Transient para que hibernate no trate de mapearlos. Los atributos que no están en la clave primaria los mapeo de la forma normal, con @Column.

```
@Entity
@Table(name = "fincaabonoorg")
@AssociationOverrides({
    @AssociationOverride(name = "pk.finca",
            joinColumns = @JoinColumn(name = "finca")),
                @AssociationOverride(name = "pk.abonoOrganico",
            joinColumns = @JoinColumn(name = "abonoOrg"))})
public class FincaAbonoOrg implements java.io.Serializable {

    private FincaAbonoOrgId pk = new FincaAbonoOrgId();
    private double cantidad;

     public FincaAbonoOrg() {
    }

    @EmbeddedId
    public FincaAbonoOrgId getPk() {
        return pk;
    }

    @Transient
    public Finca getFinca() {
        return getPk().getFinca();
    }

    @Transient
    public AbonoOrganico getAbonoOrganico() {
        return getPk().getAbonoOrganico();
    }

    @Transient
    public boolean getFondo() {
         return getPk().isFondo();
    }

    @Column
    public double getCantidad() {
         return cantidad;
    }
```

Como he nombrado anteriormente tuve que crear una clase FincaAbonoOrgId con los campos de la clave primaria que son abonoOrganico, Finca y si es de fondo o de cobertera. Como se puede ver en el código inferior esta tabla apunta a abonoOrganico con una relación ManyToOne

y a finca con una relación ManyToOne. A esta clase se le pone la notación @Embeddable para que pueda ser incrustado en otras clases.

```java
@Embeddable
public class FincaAbonoOrgId implements java.io.Serializable {

    private AbonoOrganico abonoOrganico;
    private Finca finca;
    private boolean fondo;

     @ManyToOne
    public AbonoOrganico getAbonoOrganico() {
        return abonoOrganico;
    }

    public void setAbonoOrganico(AbonoOrganico abonoOrganico) {
        this.abonoOrganico = abonoOrganico;
    }

    @ManyToOne
    public Finca getFinca() {
        return finca;
    }
```

- Fichero de configuración de hibernate

También tuve muchos problemas con el fichero de configuración del ORM, el persistance.xml. El mayor problema fue que al ejecutar el programa, como no estaba perfectamente mapeado, en vez de dar un error modificaba las tablas de las bases de datos. Tras investigar descubrí que esto se debía a que en el fichero de propiedades de hibernate, que es el persistance.xml, tenía la propiedad hibernate.hbm2ddl.auto en create.

Esta propiedad según cual sea su valor valida o exporta el squema a la base de datos. Como su valor era create lo que hacía era crear el squema, destruyendo todos los datos anteriores.

```
 <property name="hibernate.hbm2ddl.auto" value="create" />
properties>
```

Por lo tanto, tuve que modificar esa propiedad a validate para que validase el squema pero no modificase las tablas de la base de datos.

```
<property name="hibernate.hbm2ddl.auto" value="validate" />
```

- Problemas @ManytoMany

Otro problema relacionado con ORM que tuve es que me daba un error relacionado con los foreign key pero no me informaba de dónde en concreto estaba el error.

```
Caused by: java.lang.NullPointerException
        at org.hibernate.boot.internal.InFlightMetadataCollectorImpl.processFkSecondPassesInOrder(InFlightMetadataCollectorImpl.java:1678)
        at org.hibernate.boot.internal.InFlightMetadataCollectorImpl.processSecondPasses(InFlightMetadataCollectorImpl.java:1628)
        at org.hibernate.boot.model.process.spi.MetadataBuildingProcess.complete(MetadataBuildingProcess.java:278)
        at org.hibernate.jpa.boot.internal.EntityManagerFactoryBuilderImpl.metadata(EntityManagerFactoryBuilderImpl.java:770)
        at org.hibernate.jpa.boot.internal.EntityManagerFactoryBuilderImpl.build(EntityManagerFactoryBuilderImpl.java:797)
        at org.hibernate.jpa.HibernatePersistenceProvider.createEntityManagerFactory(HibernatePersistenceProvider.java:58)
        ... 4 more
```

Finalmente, comentando diferentes fragmentos de código, descubrí que se debía a que en los @ManyToOne debía añadir un targetEntity señalando a la clase a la que hacía referencia. El target entity indica cual es la clase a la que apunta la asociación.

```
@ManyToOne(targetEntity = ali.model.Agricultor.class)
@JoinColumn(name = "agricultor")
private Agricultor agricultor;
```

- Problema de la relación @ManyToMany

Cuando probé a insertar una finca tuve muchos problemas ya que si la intentaba insertar asociándole un agricultor no me lo insertaba en la tabla intermedia fincaagricultor. Esto se debía a que yo en un principio pensaba que en este tipo de relación daba igual quién fuese el lado propietario (el que no tiene el mapped by), sin embargo no es así.

Para entenderlo mejor lo voy a explicar con el ejemplo concreto de finca y agricultor. En un principio era agricultor el propietario. Por lo tanto, si añadía una finca y le asociaba agricultores, no se introducía en la tabla intermedia fincaagricultor. Por lo tanto, hay que definir como el lado propietario aquel al cual al crearlo en un programa le vayas a asociar el otro. Es decir, yo en el programa al introducir una finca le indico al cliente que seleccione los agricultores con los que está asociada la finca, por eso finca debe ser la clase propietaria.

```
@ManyToMany(cascade = {javax.persistence.CascadeType.REMOVE, javax.persistence.CascadeType.REFRESH})
@JoinTable(name = "fincaagricultor", joinColumns = {
    @JoinColumn(name = "finca", referencedColumnName = "idFinca")}, inverseJoinColumns = {
    @JoinColumn(name = "agricultor", referencedColumnName = "nombreUsuario")})
private List<Agricultor> agricultores;
```

```
@ManyToMany(cascade = {javax.persistence.CascadeType.REMOVE, javax.persistence.CascadeType.REFRESH},
    mappedBy = "agricultores")
private List<Finca> fincas;
```

- Clase padre herencia

Como abono orgánico y abono mineral tienen varios campos en común tenía sentido que crease una clase padre denominada abono. Para ello, necesitaba que las columnas que tenían en común se denominasen igual. La mayor parte de campos ya se denominaban igual, pero uno de los que tuve que modificar es el campo de la clave primaria de abono mineral y de abono orgánico. El problema estaba en que como hay claves extranjeras que apuntaban a esta tabla no me permitía modificarlo.

Para deshabilitar las foreign keys probé haciéndolo con SET FOREIGN_KEY_CHECKS=0; pero no funcionaba. Así que leí la documentación de MySQL [1] y decía que había que eliminar las foreignKeys y volverlas a crear.

A abonoMineral le hacían referencia cultivoabonomin y fincaabonomin. A abonoOrganico le hacían referencia eficienciaabonoorganico y fincaabonoorg. Así que las tuve que eliminar y volverlas a crear.

Para mapear la clase abono le tuve que añadir @MappedSuperclass pero el resto lo mapeé como una clase normal. En abono mineral y orgánico sólo tuve que añadir extends abono, como en todas las herencias de java, y mapear las columnas que no estaban en abono.

```
@MappedSuperclass
public class Abono implements Serializable{
```

```
@Entity
public class AbonoMineral extends Abono {
```


[1] https://dev.mysql.com/doc/refman/5.7/en/create-table-foreign-keys.html

CAPA DE LÓGICA DE NEGOCIO

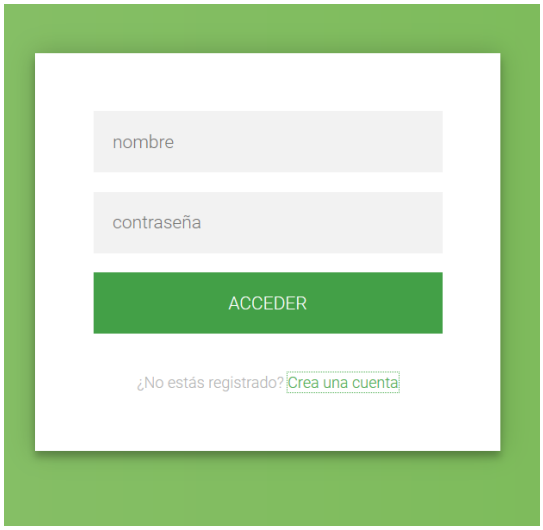
Como ya he nombrado previamente en el análisis tecnológico, voy a emplear la arquitectura Modelo-Vista-Controlador.

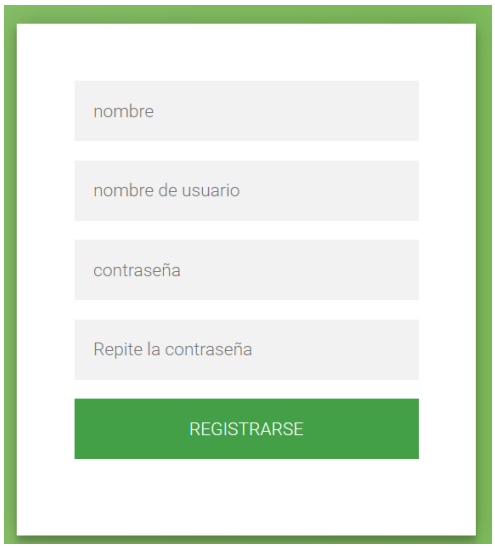
Previamente ya había creado toda la capa de persistencia. Así que me quedaba por hacer la capa de lógica de negocio y la capa de presentación. Para la lógica de negocio creé una clase gestorBD en la cual voy añadiendo los diferentes métodos que necesito.

Para trabajar con la vista he empleado servlets y jsp. Un servlet tiene varios métodos entre los que destacan el método doGet y el método doPost. Normalmente un servlet está asociado a una vista, pero puede ser que un servlet tenga varias vistas. Con el método doGet el servlet le manda a la vista los datos que esta necesita mostrar. Con el método doPost recoge peticiones y datos de la vista.

Como puse en la metodología, para ir enseñándole al cliente diferentes funcionalidades, fui haciendo a la vez la lógica de necio y la presentación a la vez, parte por parte. Además hacerlo de esta forma me ayudó a saber qué métodos debía incluir en la lógica de negocio.

Lo primero que implementé fue el login y que un agricultor se dé de alta en la aplicación. Para ello no me surgió ningún problema.





Para no guardar la contraseña tal cual hice un hash de esta empleando la librería commons-codec-1.10.jar**.** En concreto empleé un hash sha-1.

```
String pwdEn=DigestUtils.sha1Hex(pwd);
Agricultor a = new Agricultor(usuario,pwdEn,nombre);
```

Lo bueno del hash es que a partir de este no se puede obtener el texto decodificado. Además para un mismo texto siempre se obtiene el mismo hash por lo tanto se puede comparar para ver si la es la misma contraseña.

```
Agricultor a = em.find(Agricultor.class, usuario);
String pwdEn=DigestUtils.sha1Hex(contrasenia);
if (pwdEn.equals(a.getContrasenia())) {
    return true;
}
```

Otro aspecto interesante de un hash es que para entradas parecidas obtiene salidas totalmente diferentes. Y el hash es siempre de la misma longitud por lo tanto no aporta detalles sobre la longitud de la contraseña. En este ejemplo la contraseña era 1234.

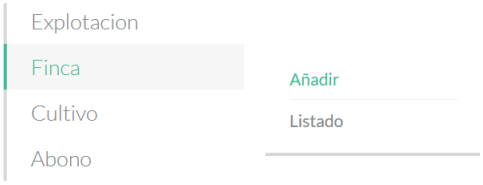
	nombreUsuario	contrasenia	nombre
	alicia	7110eda4d09e062aa5e4a390b0a572ac0d2c0220	alicia beriain

A continuación tuve que añadir algo que también está estrechamente relacionado con la seguridad de la aplicación. Muchas veces se puede pensar que si a un cliente le mostramos únicamente un menú y este sólo le va a mostrar sus datos, no podrá acceder a los de otros agricultores. Pero esto no es cierto, ya que pueden acceder manipulando la url. Por lo tanto, a todos los servlets a los que sólo debe acceder el agricultor logueado les definí su url con /agricultor delante. Para ello creé un filtro que afecta a todos los servlets que tengan su url definida con /agricultor delante. Este filtro si una persona no está dada de alta en la aplicación o no puede acceder a esa información lo reenvía a la página de login.

```
public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse resp = (HttpServletResponse) response;
    HttpSession sesion = req.getSession();
    Agricultor agricultor = (Agricultor) sesion.getAttribute("agricultor");
    if (agricultor == null) {
        String query = req.getQueryString();
        String returnURL;
        if(query==null){
            returnURL = req.getRequestURL().toString();
        }else{
            returnURL = req.getRequestURL() + "?" + req.getQueryString();
        }
        sesion.setAttribute("returnURL", returnURL);
        resp.sendRedirect(req.getContextPath() + "/index.jsp");
    } else {
        chain.doFilter(request, response);
    }
}
```

Después de implementar todo lo relacionado con el login y el registro diseñé el menú que se va a mostrar una vez se ha accedido a la aplicación. En este se muestran las opciones de administración de explotación, finca, cultivo y abono.

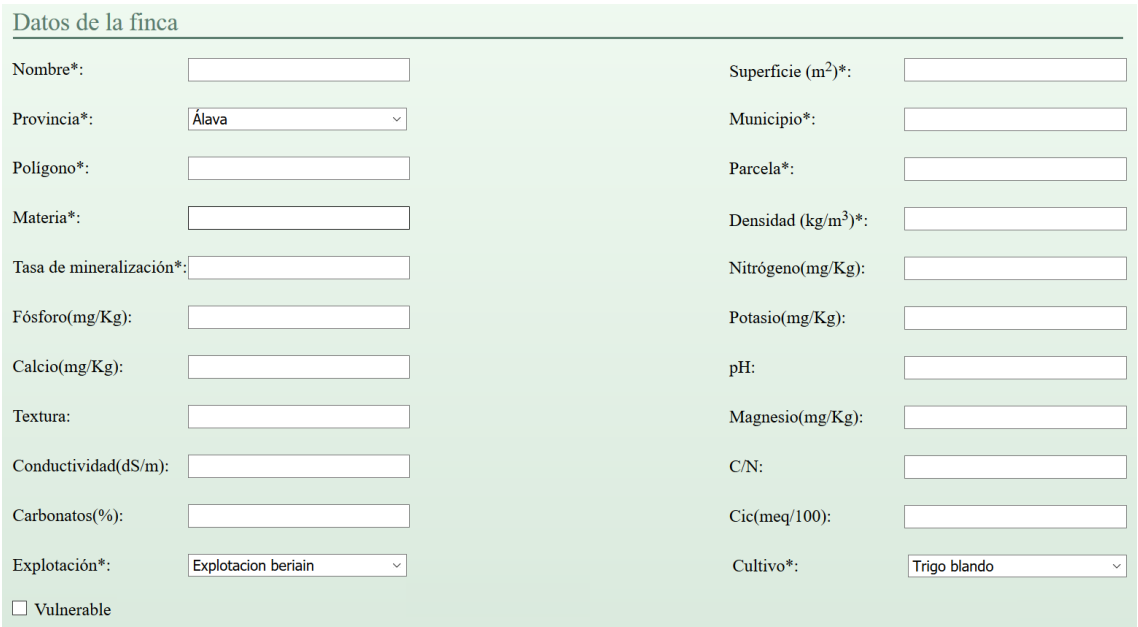
OPERACIONES CON FINCAS



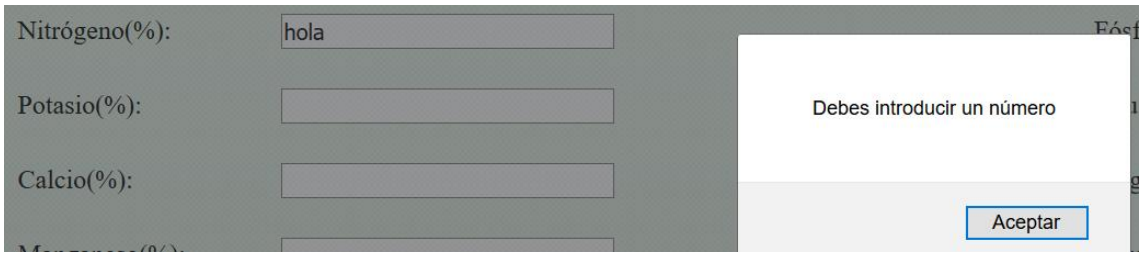
A la hora de programar el añadir una nueva finca sí que tuve un problema y es que en una finca hay muchos campos que no es necesario que se rellenen, depende de si el agricultor quiere dar más detalles o no. Por lo tanto, para no tener problemas a la hora de parsear a doubles hice un if en el que comprobaba que si la propiedad estaba vacía ponía el valor a 0.

```
String paraje=request.getParameter("paraje");
double superficie= !request.getParameter("superficie").isEmpty() ? Double.parseDouble(request.getParameter("superficie")) : 0;
double nitrogeno= !request.getParameter("nitrogeno").isEmpty() ? Double.parseDouble(request.getParameter("nitrogeno")) : 0;
double potasio=!request.getParameter("potasio").isEmpty() ? Double.parseDouble(request.getParameter("potasio")) : 0;
double fosforo=!request.getParameter("fosforo").isEmpty() ? Double.parseDouble(request.getParameter("fosforo")) : 0;
Provincia provincia=!request.getParameter("provincia").isEmpty() ? gbd.getProvincia(request.getParameter("provincia")) : null;
```

En el html en los capos que eran obligatorios les puse que fueran de tipo required. Es decir, no permite hacer clic en el botón añadir finca hasta que están seleccionados.



En aquellos campos en los que se debe introducir un número lo he comprobado mediante JavaScript. Para ello si el valor del input es NaN (Not-a-Number) sale un alert indicando que se debe introducir un número y se borra el valor del input.




```
window.onload = function () {
    var numeros = document.getElementsByClassName("num");
    for (var i = 0; i < numeros.length; i++) {
        numeros[i].onblur = selOrganico;
    }
};
function selOrganico(evento) {
    var evt = evento || window.event;
    var target = evt.target || evt.srcElement;
    var i = target.value;
    if (isNaN(i)) {
        alert("Debes introducir un número");
        target.value=null;
    }
}
```

Para permitir al cliente seleccionar una finca programé que se mostrase un listado de todas las fincas, y que permitiese al hacer clic en el nombre seleccionar una de ellas.

Listado de sus fincas

Listado de fincas

Eliminar	Nombre	Provincia	Municipio	Polígono	Parcela	Superficie
	Finca Trigo	Álava	56.0	36.0	45.0	123.0
	Finca Cebada	Rioja, La	8.0	14.0	67.0	300.0
	Finca Los Olivos	Rioja, La	14.0	11.0	4.0	500.0

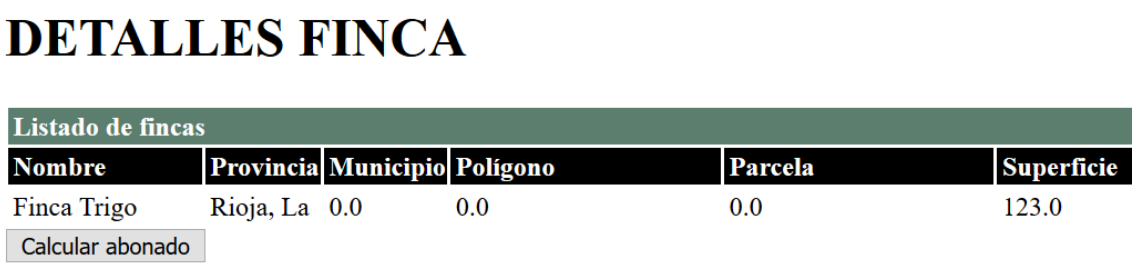
En el listado de las fincas a la izquierda aparece un símbolo que permite eliminar una finca. Al intentar eliminar una finca me surgía este problema:

```
INFO: HHH000229: Running schema validator
java.lang.IllegalArgumentException: Removing a detached instance ali.model.Finca#1
        at org.hibernate.jpa.event.internal.core.JpaDeleteEventListener.performDetachedEntityDeletionCheck(JpaDeleteEventListener.java:52)
        at org.hibernate.event.internal.DefaultDeleteEventListener.onDelete(DefaultDeleteEventListener.java:89)
        at org.hibernate.event.internal.DefaultDeleteEventListener.onDelete(DefaultDeleteEventListener.java:56)
        at org.hibernate.internal.SessionImpl.fireDelete(SessionImpl.java:920)
        at org.hibernate.internal.SessionImpl.delete(SessionImpl.java:859)
        at org.hibernate.jpa.spi.AbstractEntityManagerImpl.remove(AbstractEntityManagerImpl.java:1179)
```

Esto se debía a que al hacer el find finca fuera de la función y pasarle la finca, le estaba pasando una finca desenlazada (detached), porque el find se hacía en un contexto de persistencia distinto al del delete [1]. Para solucionarlo tuve que hacer el find y el remove en la misma función.

[1] https://stackoverflow.com/questions/2428706/jpa-thinks-im-deleting-a-detached-object

Cuando selecciona una finca del listado aparecen los detalles de la finca y un botón que permite calcular el abonado.

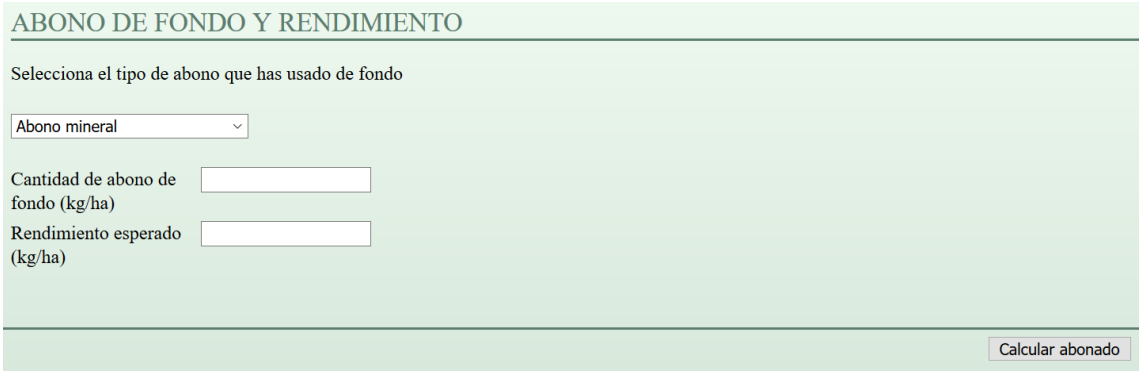


Calculadora abono

Esta es una de las partes que me parecieron más difíciles de programar. Hablando con el cliente quedamos en que para calcular la cantidad de abono lo mejor era añadir un botón cuando se veía el detalle de la finca.

Como he explicado en el apartado "CÁLCULOS QUE REALIZA LA APLICACIÓN" lo primero que debe hacer el agricultor es seleccionar el tipo de abono de fondo que ha utilizado, la cantidad que ha añadido de ese abono y el rendimiento que espera obtener de ese cultivo.

El mayor problema era cómo elegir el tipo de abono que quería. Para ello hice una pantalla en en la que permite elegir entre los dos tipos de abonos e introducir la cantidad de abonos en (kg/ha) y el rendimiento esperado (kg/ha).



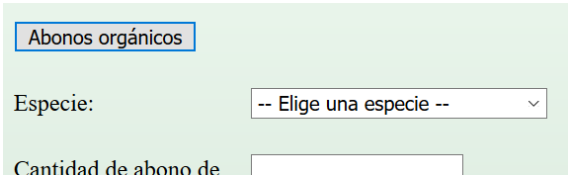
Según si selecciona abono orgánico o mineral aparecen unos desplegables u otros. Para elegir un abono orgánico no podía poner simplemente un desplegable ya que estos abonos no tienen un nombre que los defina. El cliente me indicó que el usuario los debía elegir según las siguientes características: la especie, el tipo, la procedencia, la actividad de la granja, el alojamiento, el bebedero y unas características adicionales.

Otro problema era que los abonos orgánicos que me había pasado el cliente tenían sólo algunas de las características. Sin embargo había tres características que todos los abonos tenían. Estas eran la especie, el tipo y la procedencia. Por lo tanto, le propuse al cliente hacer tres desplegables en los que permitiera elegir una de estas tres características y que posteriormente con estos filtros mostrase una tabla.

Para ello mediante Javascript hice que cuando se selecciona Abonos orgánicos, por si acaso el usuario había seleccionado antes los abonos minerales, desaparece el desplegable de abonos minerales y pone su valor al de por defecto. Además aparece un desplegable para la especie.

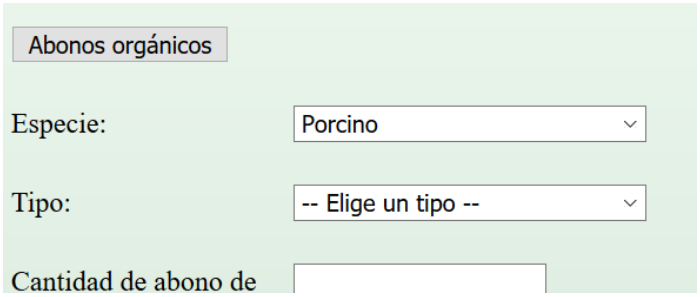
```
window.onload = function () {
    var abono = document.getElementById("abono");
    abono.onchange = deplegable;
};

function deplegable(evento) {
    var evt = evento || window.event;
    var target = evt.target || evt.srcElement;
    var valor=target.value;
    var aux = valor.localeCompare("abonoorganico");
    if (aux === 0) {
        mostrarEspecie(evento);
    } else {
         mostrarAbMin(evento);
    }
}
```

```
function mostrarEspecie(evento) {
    var evt = evento || window.event;
    var target = evt.target || evt.srcElement;
    if (evento)
        evento.preventDefault();
    else
        window.event.returnValue = false;

    var divBotMin = document.getElementById("divBotMin");
    divBotMin.style.display = "none";
    var especies = document.getElementById("divEsp");
    especies.style.display = "block";
    especies.onchange = mostrarTipo
}
```



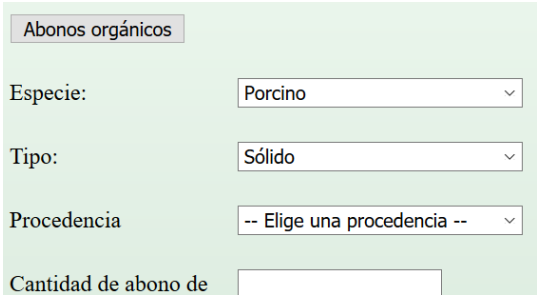
Al seleccionar una especie hice que apareciera un desplegable con los diferentes tipos. En el caso que se seleccione otra especie en el desplegable anterior los valores del desplegable tipo se modifican. Para ello tuve que emplear Javascript y AJAX. Con AJAX llamé al servlet “TiposEspecie” que según la especie me devolvía un array de Strings con los posibles tipos. Posteriormente los introducía en el select. Estos datos los tuve que obtener con un Ajax ya que al obtenerse en tiempo de ejecución el servidor no se los podía mandar.

```
function mostrarTipo(evento) {
    var evt = evento || window.event;
    var target = evt.target || evt.srcElement;
    if (evento)
        evento.preventDefault();
    else
        window.event.returnValue = false;

    var tipos = document.getElementById("divTipo");
    tipos.style.display = "block";
    var especies = document.getElementById("especie");
    $.ajax("TiposEspecie", {
        data: {'especie': especies.value},
        dataType: 'json',
        async: false,
        cache: false,
        success: function (data) {
            var tipos = document.getElementById("tipo");
            tipos.innerHTML = '<option disabled="" selected="" value=""> -- Elige un tipo -- </option>';
            for (i = 0; i < data.length; i++) {
                tipos.innerHTML = tipos.innerHTML + '<option value="' + data[i] + '">' + data[i] + '</option>';
            }
            tipos.onchange = mostrarProcedencia
        },
        error: function (xhr, status, ex) {
            alert("Error (" + xhr.status + "):" + status);
        }
    });
}
```

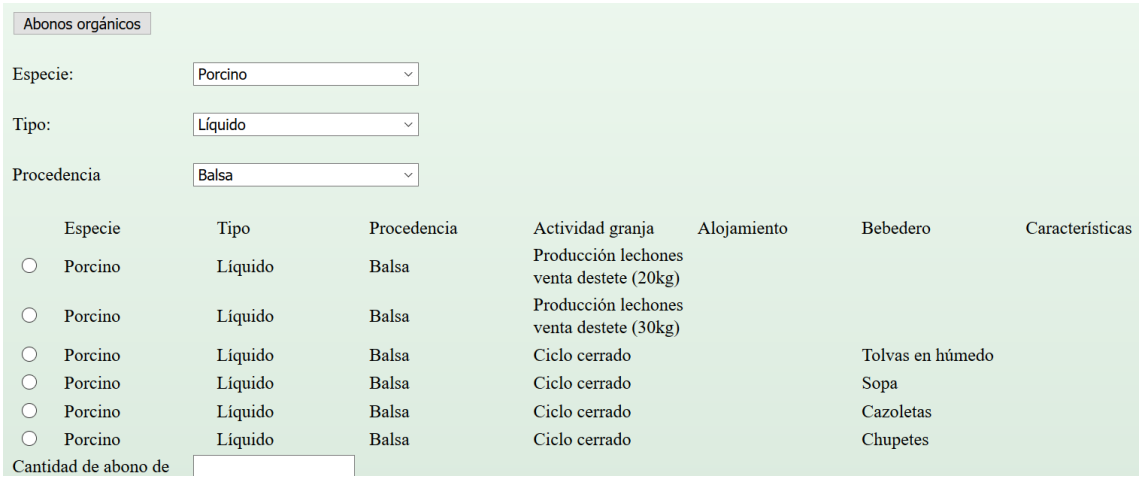


A continuación, hice que al seleccionar un tipo apareciese un desplegable que permitía seleccionar la procedencia. Lo hice de la misma forma que para tipo.

```
function mostrarProcedencia(evento) {
    var evt = evento || window.event;
    var target = evt.target || evt.srcElement;
    if (evento)
        evento.preventDefault();
    else
        window.event.returnValue = false;

    var procedencia = document.getElementById("divProc");
    procedencia.style.display = "block";
    var especies = document.getElementById("especie");
    var tipos = document.getElementById("tipo");
    $.ajax("Procedencias", {
        data: {'especie': especies.value, 'tipo': tipos.value},
        dataType: 'json',
        async: false,
        cache: false,
        success: function (data) {
            var procedencias = document.getElementById("procedencia");
            procedencias.innerHTML = '<option disabled="" selected="" value=""> -- Elige una procedencia -- </option>';
            for (i = 0; i < data.length; i++) {
                procedencias.innerHTML = procedencias.innerHTML + '<option value="' + data[i] + '">' + data[i] + '</option>';
            }
            procedencias.onchange = mostrarTabla;
        },
        error: function (xhr, status, ex) {
            alert("Error (" + xhr.status + "):" + status);
        }
    });
}
```



Finalmente, con Javascript y AJAX hice que al seleccionar una procedencia se mostrase una tabla con los abonos que cumplían esos tres filtros. Además de estas tres características en la tabla también se muestran: la actividad de la granja, el alojamiento, el bebedero y unas características adicionales. De esta manera el agricultor puede seleccionar el abono orgánico adecuado.




```
function mostrarTabla(evento) {
    var procedencias = document.getElementById("procedencia");
    var especies = document.getElementById("especie");
    var tipos = document.getElementById("tipo");
    $.ajax("AbonosOrgETP", {
        data: {'especie': especies.value, 'tipo': tipos.value, 'procedencia': procedencias.value},
        dataType: 'json',
        async: false,
        cache: false,
        success: function (data) {
            var tabla = '<colgroup><col width="4 % "> <col width="13 % "><col width="13 % "><col width="14 % ">' +
                '<col width="14 % "><col width="14 % "><col width="14 % "><col width="14 % "> </colgroup>' +
                '<tr style="text-align: left"><td></td><td>Especie</td><td>Tipo</td><td>Procedencia</td><td>Actividad granja</td><td>Alojamiento
            for (i = 0; i < data.length; i++) {
                if (i % 2 == 0) {
                    tabla += '<tr class="par" style="text-align: left">';
                } else {
                    tabla += '<tr style="text-align: left">';
                }
                tabla += '<td> <input type="radio" name="aboRadio" value='+data[i].id+'></td>' +
                        '<td id=e' + i + '>' + data[i].especie + '</td>' +
                        '<td id=t' + i + '>' + data[i].tipo + '</td>' +
                        '<td id=p' + i + '>' + data[i].procedencia + '</td>' +
                        '<td id=ag' + i + '>' + data[i].actividadGranja + '</td>' +
                        '<td id=al' + i + '>' + data[i].alojamiento + '</td>' +
                        '<td id=b' + i + '>' + data[i].bebedero + '</td>' +
                        '<td id=c' + i + '>' + data[i].caracteristicas + '</td></tr>';
            }
            var tablaOrg = document.getElementById("tablaOrg");
            tablaOrg.style.display = "block";
            tablaOrg.innerHTML = tabla;

            var radios = document.getElementsByName('aboRadio');
            for (var i = 0; i < radios.length; i++) {
                radios[i].onclick = selOrganico;
            }
        },
        error: function (xhr, status, ex) {
            alert("Error (" + xhr.status + "):" + status);
        }
    });
```

Como esta pantalla es un form para poderle pasar los datos de los abonos al método POST del servlet decidí guardar el id en un input que no es visible.

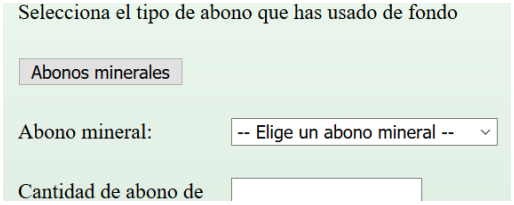
```
<div class="field" style="display: none;">
    <input type="text" name="inEsp" id="inEsp" size="30"/>
</div>
```

```
function selOrganico(evento) {
    var evt = evento || window.event;
    var target = evt.target || evt.srcElement;
    var i = target.value;
    var inid = document.getElementById("inEsp");
    inid.value = i;
}
```

Si al principio haces clic en el botón de abonos minerales desaparecen todos los desplegables de abono orgánico y aparece un desplegable con los nombres de los abonos orgánicos. Con estos abonos no tenía tantos problemas ya que tienen un nombre descriptivo, por lo tanto se pueden poner los nombres en un desplegable. Además, los abonos minerales están asociados a un tipo de cultivo y pueden ser o no de fondo. Por lo tanto, creé una función que me filtrase los abonos

de fondo que sólo sirven para el cultivo plantado en la finca. Además en un inicio sólo hay quince abonos en la aplicación y filtrándolo se reduce, por lo tanto el desplegable no queda muy largo.

```
tatic List<AbonoMineral> listaAbonosMineralesFondo(Cultivo c) throws ExcepcionDeAplicacion {
tyManagerFactory emf = null;
tyManager em = null;
tyTransaction txn = null;
{
emf = Persistence.createEntityManagerFactory("miUnidad");
em = emf.createEntityManager();
TypedQuery<AbonoMineral> query = em.createQuery("SELECT a FROM AbonoMineral a join a.cultivos c where a.aplicacionEnFondo=1 and c.id=?1", AbonoMineral.c
query.setParameter(1, c.getId());
List<AbonoMineral> l = query.getResultList();
```



Una vez elegido el abono, si se ha rellenado los kilos de abonos y el rendimiento (son "required"), permite darle al botón de calcular abonado. Este llama al método post del servlet. En el método post el servlet borra el abono de fondo que tenía la finca en la base de datos y añade el nuevo. Además también borrará el rendimiento previamente almacenado y guardará el nuevo. Después se llama a tres funciones que calculan la cantidad de nitrógeno, fósforo y potasio (NPK) que necesita añadir en el abono de cobertera para obtener el rendimiento deseado.

```
public static double calcularNitrogeno(Abono a, Double rendimiento, Double cantFondo, Finca f) {
    double extraccion = rendimiento * (f.getCultivo().getNitrogreno() / 1000);
    double aporteFondo = cantFondo * (a.getNitrogenoTotal()/100);
    double aporteSuelo = 10000 * f.getCultivo().getProfundidad() * f.getDensidad() * f.getMateria()
            * f.getTasaMineralizacion() * f.getCultivo().getContenidoN();
    double resta = extraccion - aporteFondo;
    double kgN = resta - aporteSuelo;
    return kgN;
}
```

A continuación, llama a otro servlet para pasarle las cantidades de NPK y el identificador de la finca. Este servlet permite seleccionar el abono de cobertera que se desea utilizar de la misma forma que se ha seleccionado el abono de fondo.

A continuación, el servlet llamará a una función que calcula la cantidad de abono de cobertera que se necesita añadir al cultivo.

```
public static double calcularKgAbono(Abono a, Double kgN) {
    double cantidad = 100 * (kgN / a.getNitrogenoTotal());
    return cantidad;
}
```

Además, llama a una función que borra el abono de cobertera anterior asociado a la finca y guarda el nuevo abono con la cantidad.

En el apartado de cálculos se nombra que en el caso de que falte fósforo se debería elegir un abono mineral de cobertera que no tenga nitrógeno. En los abonos minerales que me había pasado el cliente no había ningún abono que cumpliera esas condiciones. Cuando se lo comenté me dijo que la aplicación no lo tuviera en cuenta ya que el fósforo y el potasio influyen mucho menos que el nitrógeno en el rendimiento que se puede obtener de un abono. Además me dijo que es raro el caso en el que no se termina de cubrir las necesidades de fósforo y potasio.

Pero me dijo que sería interesante que la aplicación calculase la cantidad de kilogramos de fósforo y de potasio que le faltan a la finca. Por lo tanto he creado unos métodos que me

informan de la cantidad que falta de fósforo o de potasio. Para ello les paso el abono de cobertera, los kg de abono añadidos por hectárea y los del mineral que necesitan por hectárea.

```
public static double suficienteP(Abono a, Double cantidad, Double kgP){
    double cantFos = cantidad * (a.getFosforo()/100);
    if (cantFos < kgP) {
        return kgP-cantFos;
    } else {
        return 0;
    }
}
```

A continuación se llama a otro servlet encargado de mostrar el resultado. Este llama a las funciones de suficienteP y suficienteK. Este servlet también llama a las cuatro funciones correspondientes que devuelven el abono de fondo orgánico o mineral y abono de cobertera orgánico o mineral.

```
FincaAbonoMin abMinFondo= gbd.getFincaAbonoMinFondo(f);
FincaAbonoMin abMinCober= gbd.getFincaAbonoMinCobertera(f);
FincaAbonoOrg abOrgFondo= gbd.getFincaAbonoOrgFondo(f);
FincaAbonoOrg abOrgCober= gbd.getFincaAbonoOrgCobertera(f);
```

Estas funciones en el caso de que getSingleResult no encontrase ninguna fila que cumpliera las características producía una excepción de no result. Por lo tanto, tuve que poner un catch de esa excepción que no hiciera nada. Lo que hago al principio es poner el listado a null, en el caso de que salte la excepción se queda a null y finalmente se devuelve.

```
lic static FincaAbonoMin getFincaAbonoMinFondo(Finca f) throws ExcepcionDeAplicacion {
 EntityManagerFactory emf = null;
 EntityManager em = null;
 EntityTransaction txn = null;
 FincaAbonoMin abmin = null;
 try {
     emf = Persistence.createEntityManagerFactory("miUnidad");
     em = emf.createEntityManager();
     TypedQuery<FincaAbonoMin> query = em.createQuery("SELECT fam FROM FincaAbonoMin fam where fam.pk.finca.id=?1 and fam.pk.fondo=1",
     query.setParameter(1, f.getId());
     abmin = query.getSingleResult();
 }catch (NoResultException nre){
 } catch (Exception ex) {
     if (txn != null && txn.isActive()) {
```

El servlet que llama a estas funciones le pasa los resultados a una vista en la cual dependiendo de si se emplean abonos de fondo o de cobertera mostrará la información de una manera u otra.

ABONO DE FONDO QUE HAS INDICADO

Abono mineral de fondo	
Nombre	**Cantidad**
NPK 9-23-12	110.0

ABONO DE COBERTERA QUE DEBES AÑADIR

Abono mineral de cobertera	
Nombre	**Cantidad**
Urea 46	90.0

Faltan 0 Kg de Fósforo por añadir

Faltan 0 Kg de Potasio por añadir

Volviendo al tema de la vista de los detalles de la finca. Decidí que si el valor de uno de los abonos de cobertera era distinto de null se mostrase un botón que deja ver los resultados calculados la última vez. Estos datos son recuperados de la relación entre finca y los abonos. Los kg de fósforo y potasio que faltan se pueden obtener porque el rendimiento esperado se guarda en la relación entre finca y entre cultivo.

DETALLES FINCA

Listado de fincas

Nombre	Provincia	Municipio	Polígono	Parcela	Superficie
Finca Trigo	Álava	56.0	36.0	45.0	123.0

Calcular abonado

Ver resultados abonado

Finalmente como una finca un año puede tener un cultivo asociado pero al año siguiente puede tener otro, en la pantalla de detalles finca añadí un desplegable que permite cambiar el cultivo asociado a la finca.

DETALLES FINCA

Listado de fincas

Nombre	Provincia	Municipio	Polígono	Parcela
Finca Trigo	Álava	56.0	36.0	45.0

Calcular abonado

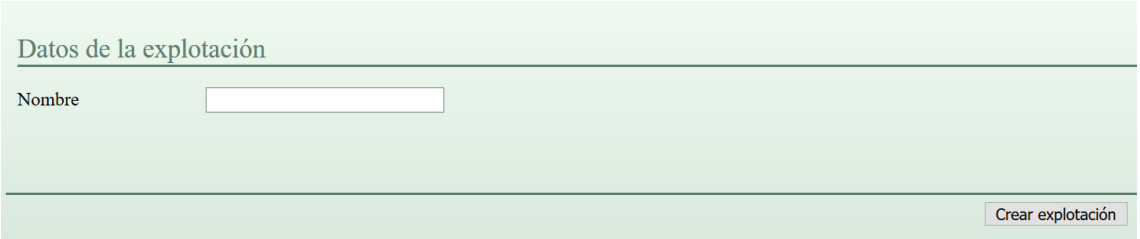
Trigo blando | Cambiar el cultivo asociado

Gracias al ORM utilizado simplemente he tenido que crear la siguiente función:

```
public static void modificarFinca(int id, Cultivo c) throws Excepc
    EntityManagerFactory emf = null;
    EntityManager em = null;
    EntityTransaction txn = null;
    try {
        emf = Persistence.createEntityManagerFactory("miUnidad");
        em = emf.createEntityManager();
        Finca f = em.find(Finca.class, id);
        em.getTransaction().begin();
        f.setCultivo(c);
        em.getTransaction().commit();
    } catch (Exception ex) {
```

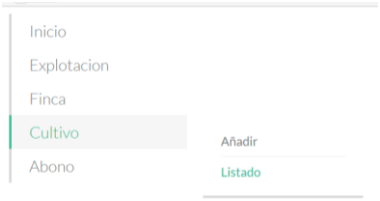
OPERACIONES EXPLOTACIÓN

Para una explotación también creé la posibilidad de añadir una nueva. Esta operación no tuvo ninguna dificultad añadida a las ya comentadas en añadir una nueva finca.



En el caso del listado de explotaciones tampoco ha tenido mucha complicación. Al hacer clic en el nombre de una explotación, se muestra un listado con las fincas en esa explotación.

OPERACIONES CULTIVO

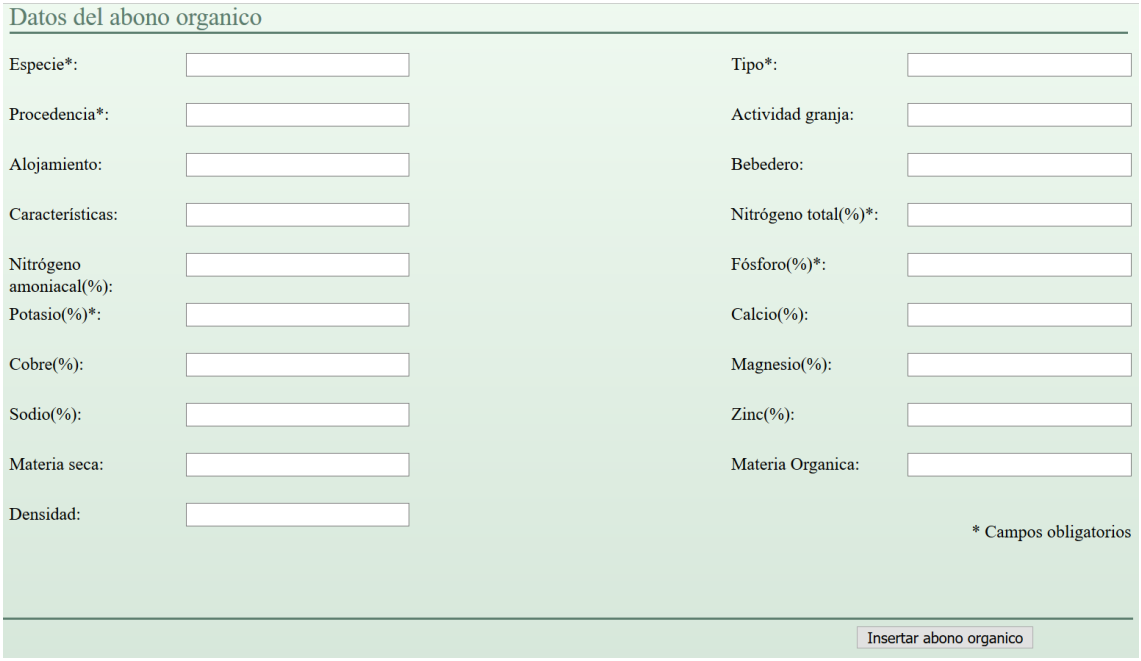


Para un cultivo también creé la posibilidad de añadir uno nuevo. Esta operación no tuvo ninguna dificultad añadida a las ya comentadas en añadir una nueva finca.

También añadí la opción de ver el listado de cultivos. En este caso se verán los que ya tenía la aplicación y los que el usuario ha añadido.

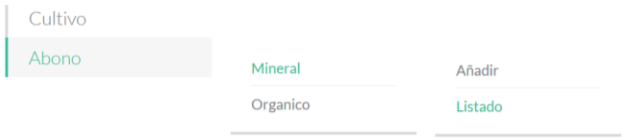
OPERACIONES ABONO ORGÁNICO

Para un abono orgánico también creé la posibilidad de añadir uno nuevo. Esta operación no tuvo ninguna dificultad añadida a las ya comentadas en añadir una nueva finca.

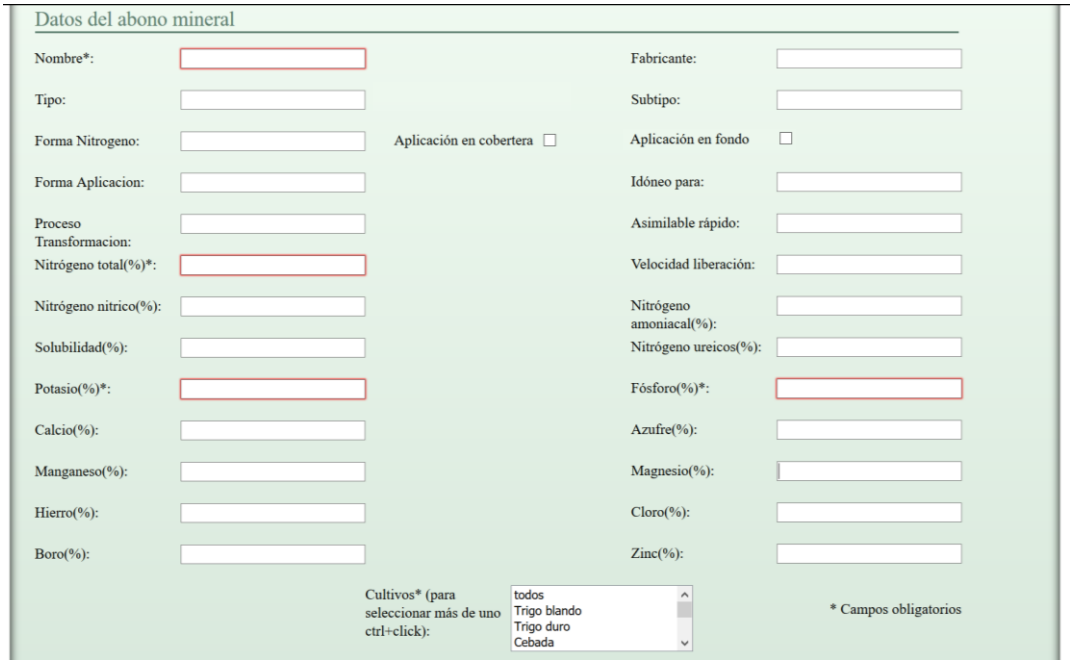


En los abonos orgánicos también he habilitado la opción de ver el listado de los abonos orgánicos. En este caso verá los abonos orgánicos que ya estaban en la aplicación más los posibles abonos que el agricultor haya añadido. Además como el listado inicial es grande tendrá la posibilidad de filtrar por especie, tipo y procedencia, como a he hecho para seleccionar el abono orgánico de fondo o de cobertera.

OPERACIONES ABONO MINERAL



He programado el añadir un nuevo abono mineral, lo he realizado de una manera similar a nueva finca.



La única complicación extra que he tenido es que un abono orgánico sólo se puede emplear en ciertos cultivos. Para ello he tenido que usar un select multiple. A este select le he añadido una option todos que permite seleccionar todos los cultivos, sin tener que ir eligiendo uno a uno.

```
<label for="cultivos">Cultivos* (para seleccionar más de uno ctrl+click):
</label>
<select name="cultivos" id="cultivos" multiple required>
    <option value="todos">todos</option>
    <c:forEach var="p" items="${cultivos}">
        <option value="${p.nombre}">${p.nombre}</option>
    </c:forEach>
</select>
```

En el servlet he añadido el siguiente código para que lo inserte correctamente:

```
String[] lista = request.getParameterValues("cultivos");
List<Cultivo> cultivos = new ArrayList<Cultivo>();
if (lista[0].equals("todos")) {
    cultivos = gbd.listaCultivos();;
} else {
    for (int i = 0; i < lista.length; i++) {
        Cultivo c = gbd.getCultivo(lista[i]);
        cultivos.add(c);
    }
}
```

En los abonos minerales también he habilitado la opción de ver el listado de los abonos minerales. En este caso verá los abonos minerales que ya estaban en la aplicación más los posibles abonos que el agricultor haya añadido.

TESTEO

Para probar los diferentes métodos como ya he dicho primero los he ido probando en la aplicación de Java. Ahí he probado con diferentes entradas y salidas para comprobar que funcionaban correctamente.

Una vez que los métodos funcionaban correctamente comprobé que los parámetros que le pasaba la aplicación web eran correctos. Es decir, poner algunos campos como obligatorios, comprobar que en los datos en los que se deben introducir números no se introduce texto.

Además le mostré la aplicación al cliente en varias ocasiones y la fue probando para poder recabar su opinión al respecto.

CAPÍTULO 4. SEGUIMIENTO Y CONTROL

A lo largo de las semanas se ha ido haciendo un seguimiento de las horas empleadas en las distintas partes del proyecto. En la siguiente tabla he realizado una comparativa entre las horas que tenía pensadas para cada uno de los apartados y las horas reales que he acabado invirtiendo.

TAREA	HORAS PROGRAMADAS	HORAS REALES	DESVIACIÓN
Memoria + Reuniones + Gestión	78	80	2,66%
Análisis de requisitos	12	16	33,33%
Análisis de casos de uso	12	12	0,00%
Cálculos aplicación	12	16	33,33%
Estudio de tecnologías	6	2	-33,33%
Diseño de bases de datos	12	12	0,00%
Diseño del programa	12	8	-33,33%
Instalación tecnologías	3	3	0,00%
Creación base de datos	15	18	20,00%
Carga de datos inicial	12	18	50,00%
Servicio web	66	100	51,51%
Interfaz gráfica	60	20	-66,66%
TOTAL	300	305	**1,66%**

Respecto a la memoria, las reuniones y la gestión la desviación es tan pequeña que prácticamente es insignificante.

En el análisis de los requisitos he tenido una desviación positiva de un 33,33%. Esto se debe a que como el tema de la ingeniería agrícola era un tema completamente nuevo para mí, tuve que reescribir varias veces los requisitos hasta que el cliente los aceptó.

Los cálculos de la aplicación me costó entenderlos un poco más de lo planificado inicialmente, esto también se debe a que eran conocimientos completamente nuevos para mí y además hay que tener mucho cuidado con las unidades que se están empleando y cómo se simplifican las unas con las otras.

En un principio pensaba que iba a tener que estudiar más las tecnologías por mi cuenta. Pero muchas ya las había visto o las estaba viendo en asignaturas de la carrera, el tiempo de estudio fue menor.

La creación de la base de datos me costó ligeramente un poco más de lo esperado. Esto se debió a que como no había trabajado previamente con MySQL tuve algunos fallos.

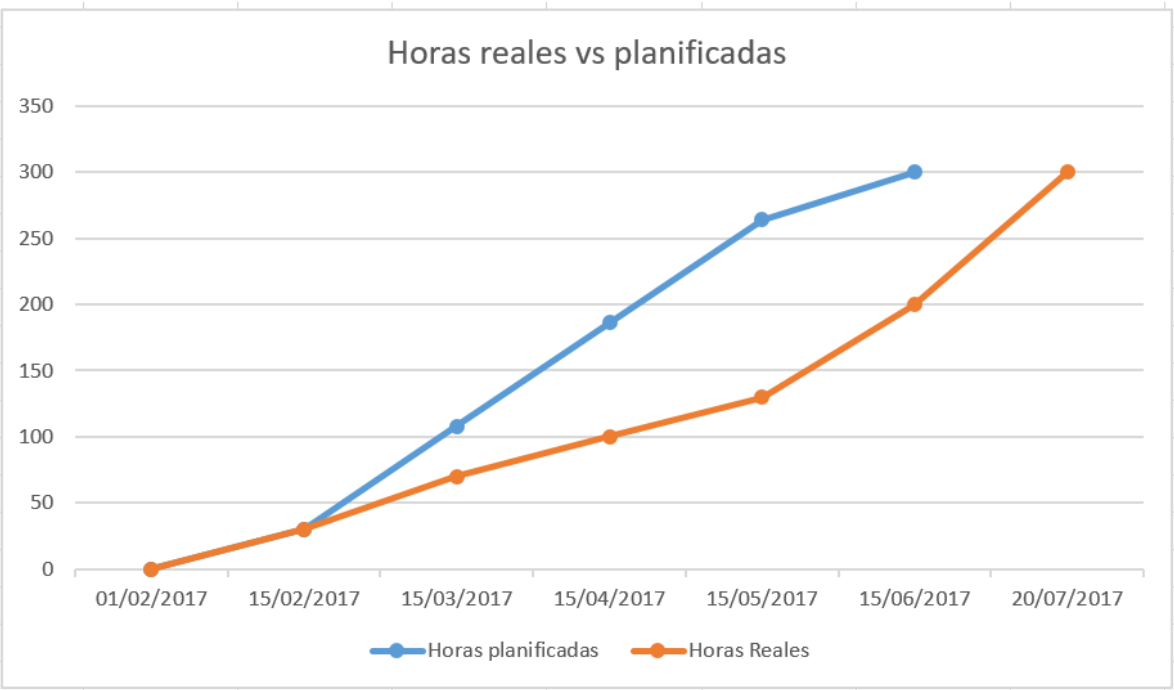
Es importante recalcar lo que me costó la carga de datos inicial. Esto se debió a dos causas, la primera es que, como he explicado anteriormente, me tardé en encontrar el método correcto para cargar la base de datos. La segunda causa es que el cliente me pasaba la información en un formato y yo posteriormente lo tenía que adaptar para poderlo introducir en la base de datos, tenía que crear la información de las tablas intermedias…

En el aspecto en el que más desviación he tenido es en la implementación de la aplicación. De esta parte en concreto la que me ha resultado más complicado son los aspectos relacionados

con el ORM. Hasta ahora no había utilizado esta tecnología y como he ido reflejando a lo largo de la memoria he tenido varios problemas enfrentándome a ella.

Como hasta este segundo semestre no había cursado la asignatura de aplicaciones web infravaloré la cantidad de horas que hay que dedicar a la implementación y pensaba que había que dedicar muchas más a la interfaz. Sin embargo, como se puede ver he acabado dedicando muchas más horas a la implementación. A pesar de que no me resulte muy complicado sí que se tarda tiempo comprobando que todo funcione correctamente y que no se produzcan excepciones.

En este gráfico se puede ver una comparativa de cómo en un principio tenía pensado entregar el trabajo el mayo, pero finalmente como no me dio tiempo lo entregué en julio.



CUMPLIMIENTO DE REQUISITOS

La aplicación ha llegado a su objetivo ya que que calcula la cantidad de abono de cobertera que hay que añadir a una finca. Sin embargo, no se han cumplido todos los requisitos porque, como ya he nombrado en el alcance, al hacerlos no se tuvo en cuanta el límite de tiempo, simplemente se pusieron todos los requisitos que el cliente me dijo. De esta forma si se quiere continuar con la aplicación no se tenían que captar de nuevos los requisitos. Me fui centrando en aquellos requisitos que el agricultor les daba prioridad.

Dentro de los requisitos funcionales no se ha cumplido del todo el punto tres ya que hay un apartado en al cuál dice que, mediante la referencia catastral, accediendo a una API del catastro, obtendremos la superficie de la finca y su nombre. Finalmente se ha hecho que sea al agricultor el que dé el nombre a la finca y ponga la superficie.

Tampoco se ha cumplido del todo el punto 6 en el apartado que dice que en el caso de que el agricultor no conozca las extracciones de nitrógeno, fósforo y potasio se calcularán. Al final el cliente no lo consideró algo importante, no me aportó las fórmulas y tampoco me aportó los datos. Sin embargo estos campos sí están creados en la base de datos por si se continúa con la aplicación.

No se ha cumplido el punto **7** de los requisitos funcionales en el cual dice que para calcular la cantidad de abono que hay que añadir a una finca hay que tener en cuenta los restos de cosecha que no se han recogido en la finca. Esto se debe a que el cliente no ha considerado que fuera tan importante, no lo ha tenido en cuanta en las fórmulas que me explicó y no me ha pasado los datos. La tabla de restos de cosecha está creada y mapeada pero no tiene datos, por lo que se podría utilizar para mejorar la aplicación en un futuro.

No se ha cumplido el punto **8** de los requisitos funcionales en el cual dice que la aplicación va a informar al agricultor como debe añadir el abonado para un cultivo en concreto. Esto se debe a que el cliente no ha considerado que fuera tan importante, no lo ha tenido en cuanta en las fórmulas que me explicó y no me ha pasado los datos. La tabla de fraccionamiento de abonado está creada y mapeada pero no tiene datos, por lo que se podría utilizar para mejorar la aplicación en un futuro. Añadir esta funcionalidad en un futuro en sería muy sencillo ya que simplemente se tendrían que mostrar los datos en un html.

No se ha cumplido el punto **10** de los requisitos funcionales en el cual dice que para calcular la cantidad de abono que hay que añadir a una finca hay que tener en cuenta si se ha añadido abono orgánico en los dos años anteriores. Esto se debe a que el cliente no ha considerado que fuera tan importante, no lo ha tenido en cuanta en las fórmulas que me explicó y no me ha pasado los datos de la tabla. La tabla de restos de cosecha está creada y mapeada pero no tiene datos, por lo que se podría utilizar para mejorar la aplicación en un futuro.

El resto de requisitos considero que sí que han sido cumplidos.

CAPÍTULO 5. CONCLUSIONES

Este trabajo me ha servido para profundizar los conocimientos aprendidos en la carrera, así como para adquirir nuevos conocimientos. Me ha resultado muy interesante el realizar una aplicación desde cero. Esto me ha ayudado a darle una visión global a los diferentes conocimientos tratados en la carrera como puede ser diseño y programación de bases de datos, ingeniería del software, programación de aplicaciones web…

A pesar de que he ampliado los conocimientos adquiridos en estas asignaturas también he aprendido a trabajar con tecnologías prácticamente nuevas para mí, como puede ser el mapeo ORM. En mi aplicación he tenido que mapear tablas y relaciones muy complicadas. Al mapear la base de datos me han surgido muchos errores, los cuales me costaba bastante localizar la razón por la que sucedían. Pero una vez hecho el mapeo resulta mucho más fácil trabajar con la base de datos.

Otra novedad del trabajo es pensar cómo realizar la interfaz. Es decir, cómo y dónde presentar los distintos botones y funcionalidades de forma que sea el sitio más lógico. Hasta ahora nunca había hecho una aplicación web en la que tenía que diseñar toda la interfaz. He tenido que pensar cómo mostrar la información, donde poner los diferentes botones, a que pantalla se debería pasar después de otra, cuando filtrar la información…

También he aprendido mucho gracias a haber trabajado por primera vez con un cliente real. He aprendido a negociar con el cliente, preguntarle para asegurarme de que he entendido todo correctamente… Es importante destacar que he a pesar de que solamente he redactado cuatro actas. He tenido comunicación “no formal” constante con el cliente ya sea en persona o por teléfono.

Como he comentado en el seguimiento y control el producto sí que cumple con su objetivo, pero no cumple todos los requisitos. Como la base de datos de muchos de esos requisitos sí se ha realizado, eso puede dar pie a que en un futuro se continúe con la aplicación y se perfeccione.

REFERENCIAS

- Stack Overflow, para resolver dudas y fallos: https://stackoverflow.com/
- El apartado de hibernate de Mkyong, para aprender a realizar el ORM: https://www.mkyong.com/hibernate/
- El apartado de hibernate de CodeJAva, para aprender a realizar el ORM: http://www.codejava.net/frameworks/hibernate
- Documentación de MySQL para dudas relacionadas con este: https://dev.mysql.com/doc/refman/5.7/en/
- WikiBook Java Persistence/JPQL para hacer las consultas: https://en.wikibooks.org/wiki/Java_Persistence/JPQL#JOIN
- Temas 2,3,4 y 5 de los apuntes de la asignatura de diseño de bases de datos.
- Tema 6 de los apuntes de la asignatura de diseño de bases de datos.
- Tema 5 y 7 de los apuntes de la asignatura de diseño tecnológico de sistemas de información.
- Todos los temas de los apuntes de la asignatura de programación de aplicaciones web.
- Tema de los hash de la asignatura de los apuntes de seguridad.